Implementation Security In Cryptography

Lecture 11: Fault Attacks

Recap

- Till the last lecture
  - Masking…

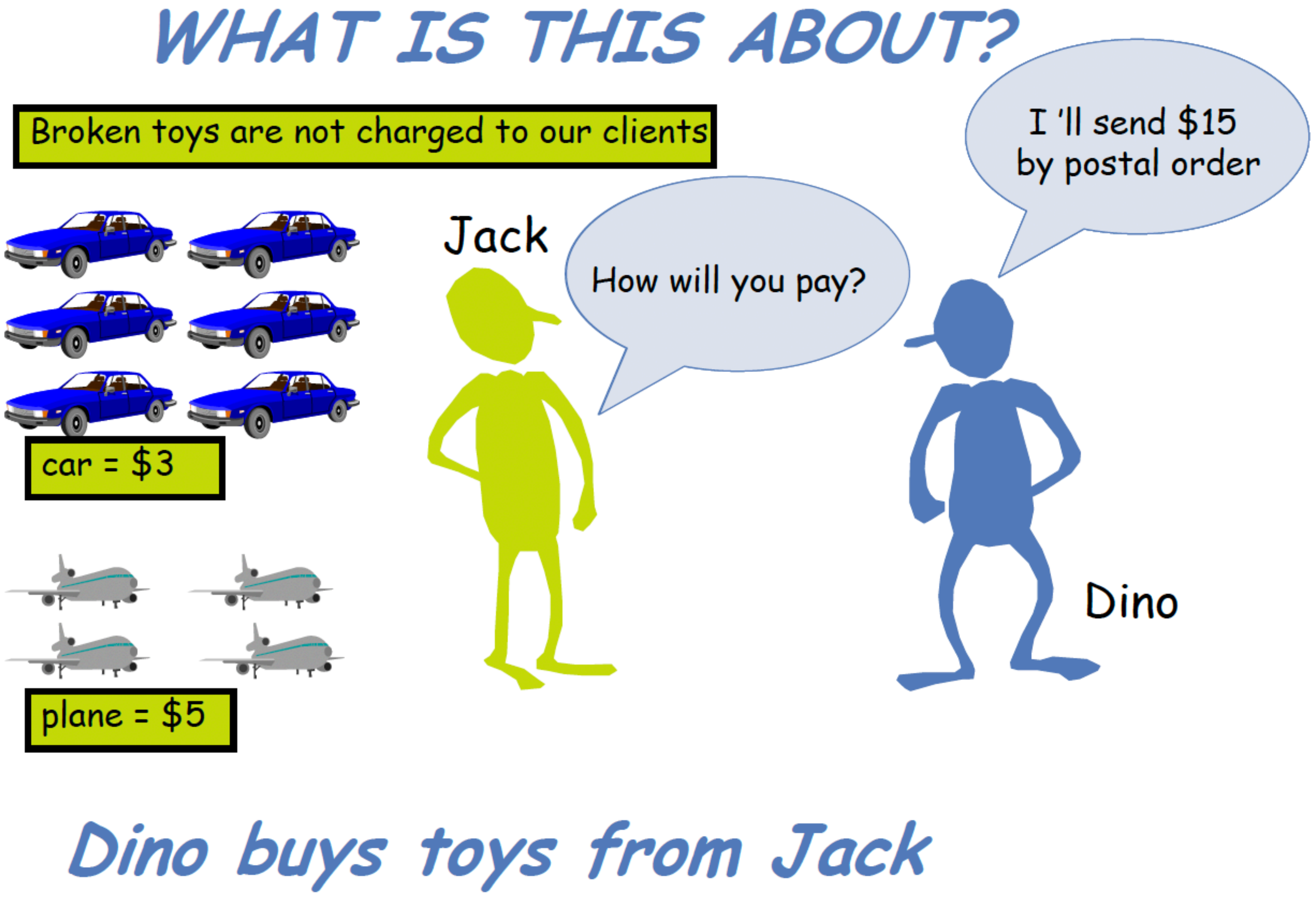
Today

- Fault Attacks

Faults in Our Life…

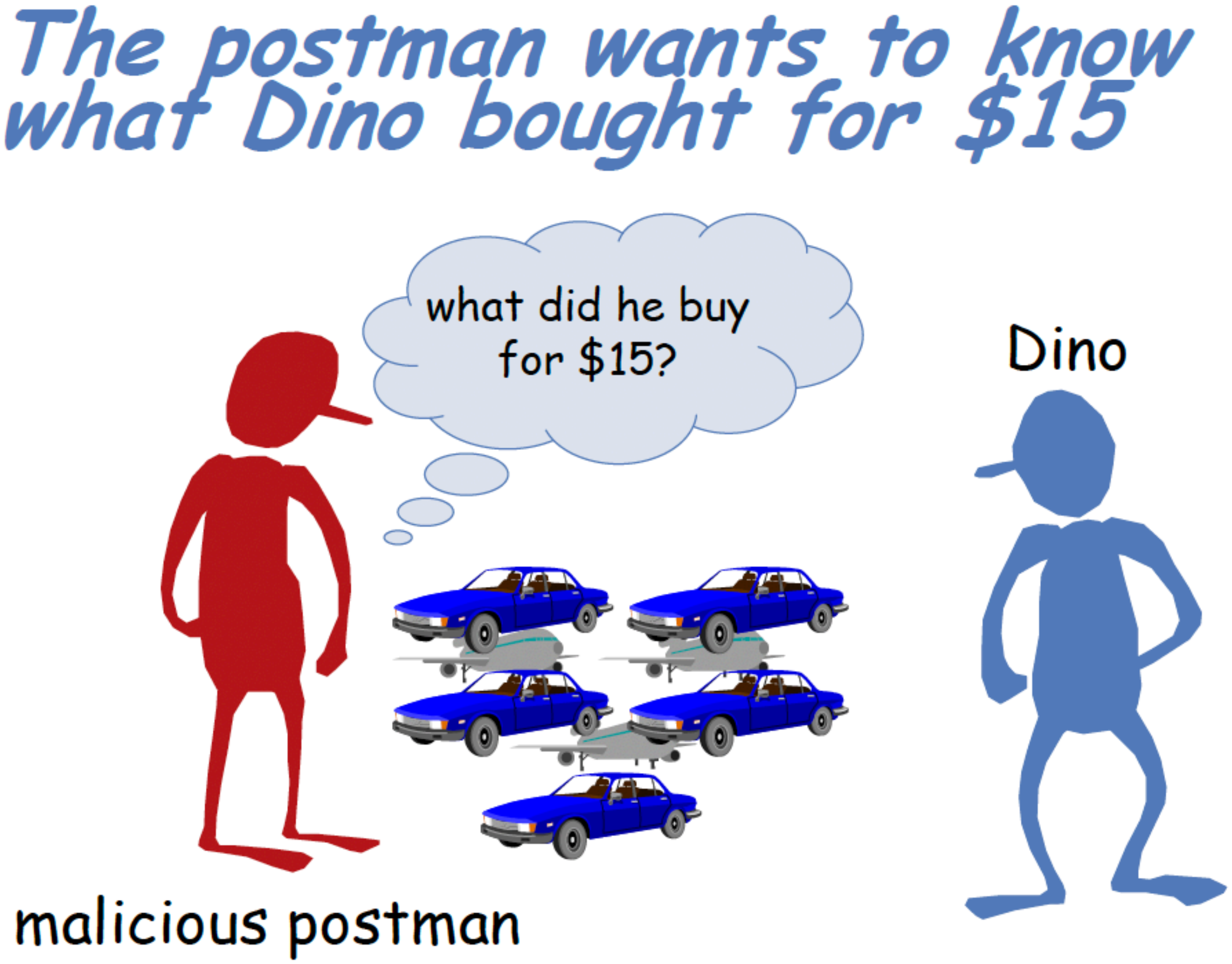
- Happens….
- We are all human being (Hope there is no AI agent in my class) :P
- We all learn from our faults….
  - The learning is what I am looking for here

Faults in Our Life…



- The Sorcerer's Apprenctice's Guide to Fault Attacks, FDTC 2006

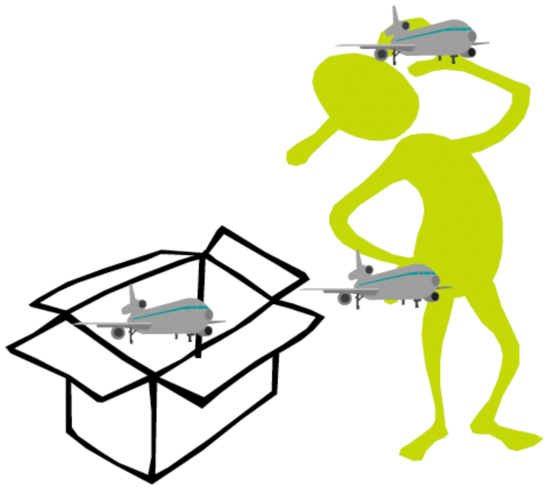
Faults in Our Life…



- The Sorcerer’s Apprenctice’s Guide to Fault Attacks, FDTC 2006

Faults in Our Life…

In the meanwhile Jack prepares the DHL



- The Sorcerer’s Apprenctice’s Guide to Fault Attacks, FDTC 2006

Faults in Our Life...

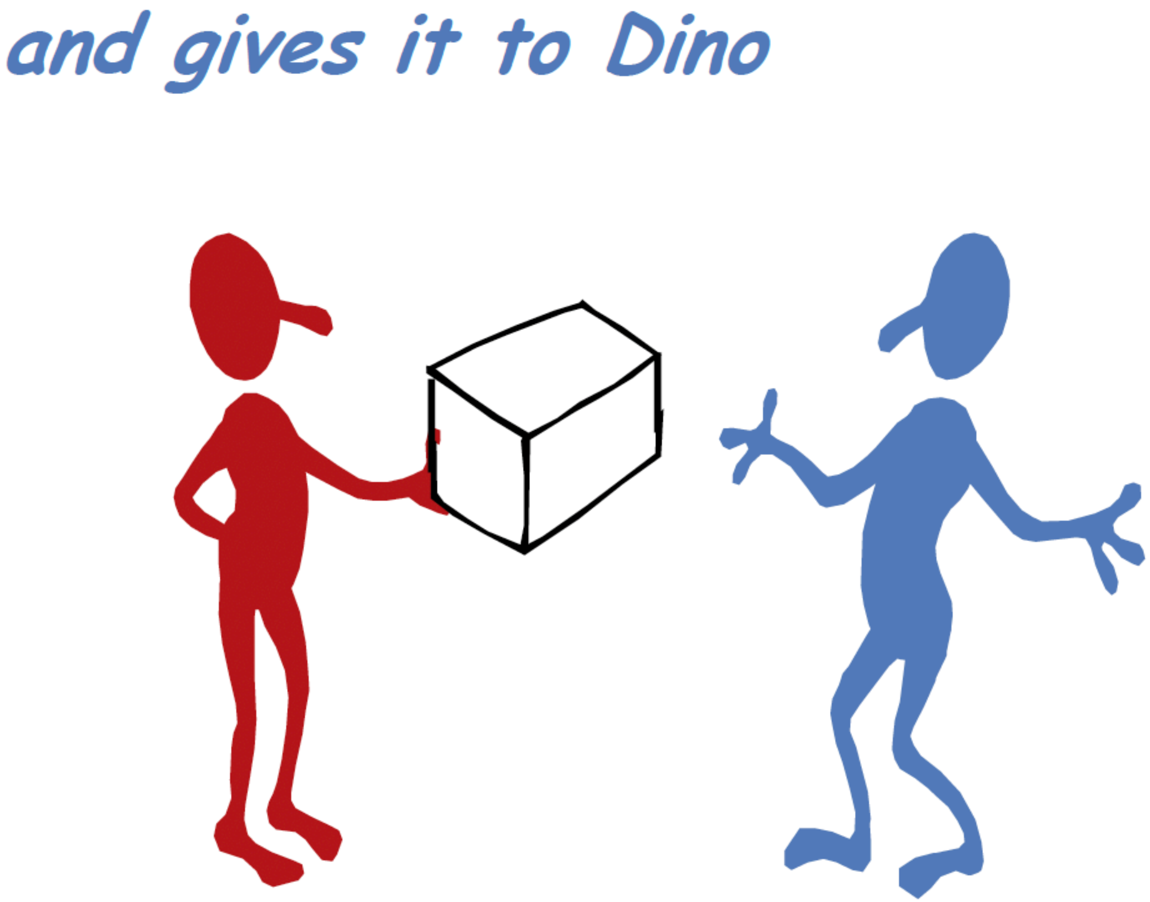
- The Sorcerer's Apprenctice's Guide to Fault Attacks, FDTC 2006

Faults in Our Life…

Who kicks it strong enough to break one toy

- The Sorcerer’s Apprenctice’s Guide to Fault Attacks, FDTC 2006

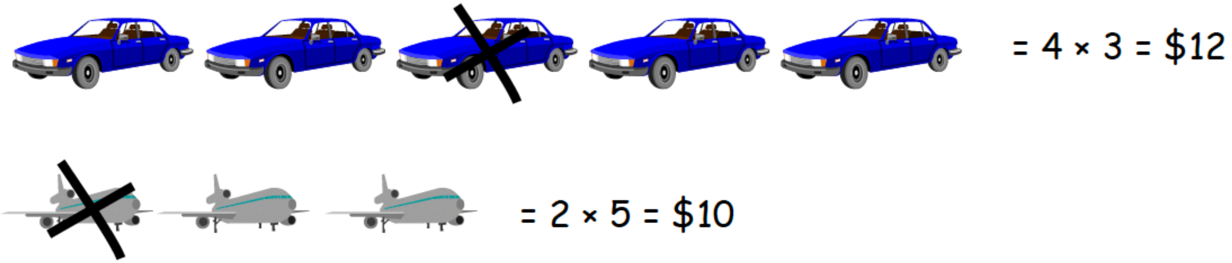
Faults in Our Life…



- The Sorcerer's Apprenctice's Guide to Fault Attacks, FDTC 2006

Faults in Our Life...

a week later he monitors Dino's postal order...



Lesson learned: **Fault attacks** can also extract secrets from tokens!

Hardware faults can have various sources: voltage glitches, light beams, laser beams...

- The Sorcerer's Apprenctice's Guide to Fault Attacks, FDTC 2006

Faults in Our Crypto…

- Introduction of faults in the normal execution of cryptographic algorithms and analysis of faulty output to obtain the key
- First conceived in 1996 by Boneh, Demillo and Lipton
- E. Biham developed Differential Fault Analysis (DFA) of DES
- Today there are numerous examples of fault analysis of block ciphers such as AES under a variety of fault models and fault injection techniques
- Popular Fault Injection Techniques – Clock Glitches, Voltage Glitches, EM and Optical Injection Techniques

What Faults are Up To?

Serious Security: Rowhammer returns to gaslight your computer

Gaslights produce a telltale flicker when nearby lamps are lit; DRAM values do something similar when nearby memory cells are accessed.

Written by Paul Ducklin

JULY 10, 2023

NAKED SECURITY | DATA LEAKAGE | ROWHAMMER | SERIOUS SECURITY

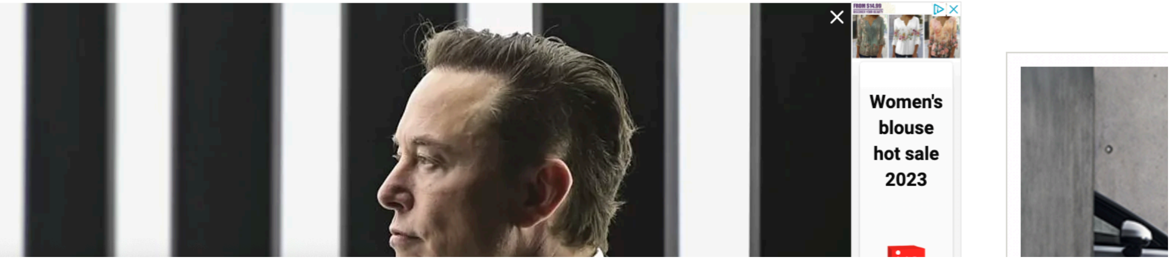
You're probably familiar with the word *gaslighting*, used to refer to people with the odious habit of lying not merely to cover up their own wrongdoing, but also to make it look as though someone else is at fault, even to the point of getting the other person to doubt their own memory, decency and sanity.

You might not know, however, that the term comes from a 1930s psychological thriller play called *Gas Light* (spoiler alert) in which a manipulative and murderous husband pretends to spend his evenings out on the town

Technology

Using just a $25 device a researcher hacked into Elon Musk's Starlink system

What will the technology mogul have to say about this?



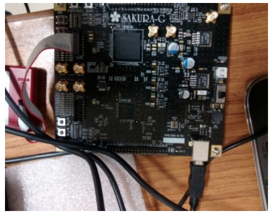
Faults in Our Crypto...

Small World

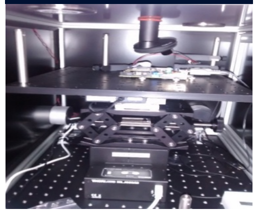
Voltage-Glitch

EM-FI

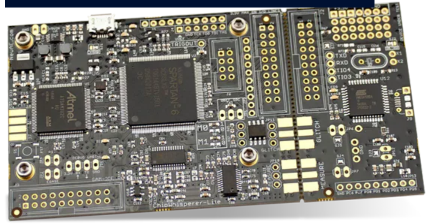
Clock-Glitch



Laser-FI



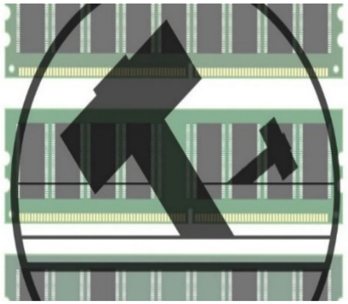
Chipwhisperer



- Tight control on injection timing
  - You can hit the variable you want at a specific instant
- Multi-bit — but can be made single-bit too
- Biased fault distribution
- Mainly transient
- Multiple injection challenging

Big World

Row-Hammer



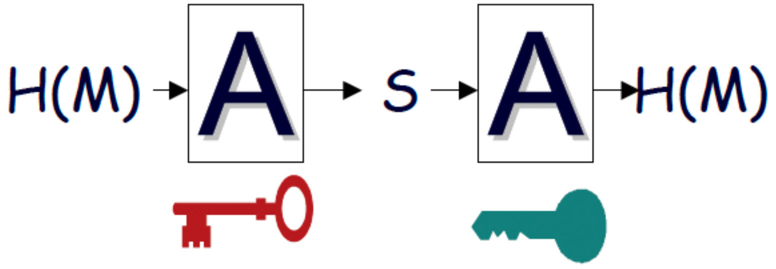
PLUNDER VOLT

CACHE WARP

- Timing can be controlled but precision is less
  - Target variable must be in RAM for quite sometime
- Single-bit — fairly well controllable
- Mainly persistent
- Multiple injection is easier

A Break From AES…RSA Signature

- Let's talk RSA
- Public key algorithm, — Encryption and Signature
    - We talk about signature.
- **Simple Idea**:
    - Alice generates a secret and a public key.
    - Gives the public key to "Public"
    - Signs a message M with secret key and generates signature C
    - Everyone with the public key can verify that:
        - The C is a valid signature of M
        - C is generated by Alice only and nobody else



RSA Signature in Brief…

- Alice generates two large primes p, q and computes $N = pq$
- It also finds e and d, where $ed \equiv 1 \ mod \ \phi(N)$, this $\phi(N)$ is called Eular's totient function and $\phi(N) = (p-1)(q-1)$, as p, q are primes.
- p, q, d is secret key
- N, e is the public key
- **Sign**: $m \in \mathbb{Z}_n^*$, compute $s = m^d \ mod \ N$, and returns $s||m$
- **Verify**: Check if $m' == m$, where $m' = s^e \mod N$
- I have omitted some crucial details here for simplicity. But that's not for what we are going to explain..
    - E.g, m is not the message but is a hash of the message..also there are some paddings needed for security…

RSA-CRT

- CRT stands for **Chinese Reminder Theorem** — check it out!!!
- RSA-CRT is a performance optimisation trick…

$$\begin{cases} a \equiv 1 \pmod{p} \\ a \equiv 0 \pmod{q} \end{cases} \text{ and } \begin{cases} b \equiv 0 \pmod{p} \\ b \equiv 1 \pmod{q} \end{cases}$$

$$d_p = d \pmod{p-1}$$
$$d_q = d \pmod{q-1}$$

$$s_p = m^{d_p} \pmod{p}$$
$$s_q = m^{d_q} \pmod{q}$$

$$s = a \times s_p + b \times s_q \pmod{N}$$

Attacking RSA-CRT…

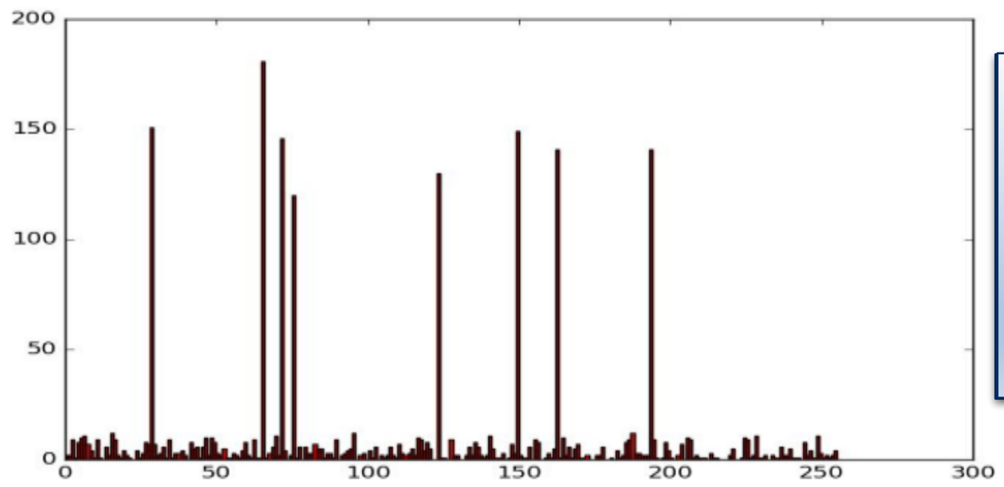
- Again, we want to find the secret key…So attack the signing process..
- Let's say, for a message m, I can repeat the signature generation process.
- What I do:
  - Generate the correct signature $s = \text{sign}(m, d)$
  - On the same message generate a faulty signature $\hat{s} = \text{sign}(m, d)$
    - Fault happens during the computation of the sign.
    - The fault only corrupts the computation of s_p or s_q, but not both

Attacking RSA-CRT…

- Let $\hat{s} = a \times s_p + b \times \hat{s}_q \ mod \ N$
- Let $s = a \times s_p + b \times s_q \ mod \ N$
- Let $\Delta = s - \hat{s} = b(s_q - \hat{s}_q)$
- Now, observe that, since $b \equiv 0 \ mod \ p$
  - $\Delta = b(s_q - \hat{s}_q)$ is divisible by p
  - So, $\Delta = kp$, and $gcd(s - \hat{s}, n) = p$
- Ok, you got p, you know N, so you know q — game over!!!
- Attack is also possible if you do not use CRT — analysis is slightly different

Let There Be Faults: Fault Model

- Key component of a fault attack
- Attack procedure changes according to the fault model
- <u>Random localized faults</u>
  - Bit/nibble/byte fault
  - Most general model
- <u>Biased faults</u>
  - Device-dependent model
- <u>Instruction skip/modify.</u>
- <u>Constant fault</u>
  - Stuck-at-0/1
- <u>Persistant fault</u>

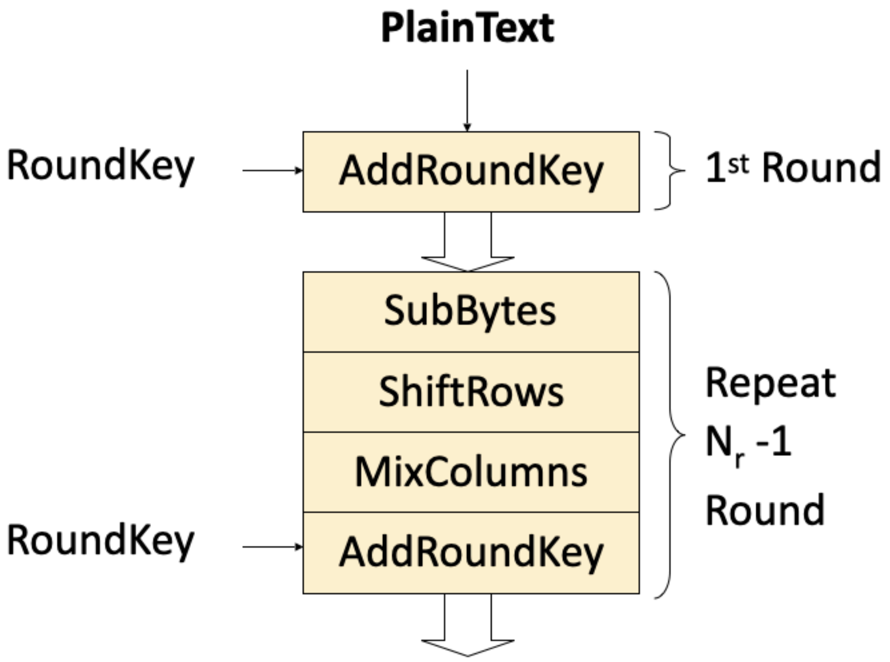


In a fault space of size 256, only 8 faults occur in 98% of the total injections!!!

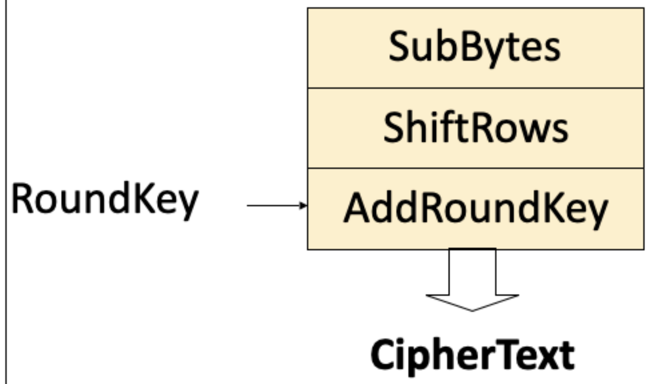
```
ldi   r1   0;
ld    r1   #M1
ldi   r2   0
ld    r2   #M2
add   r1   r2
str   r1
```

```
ldi   r1   0;
nop
ldi   r2   0
ld    r2   #M2
add   r1   r2
str   r1
```

The AES Story...



First 9 Rounds

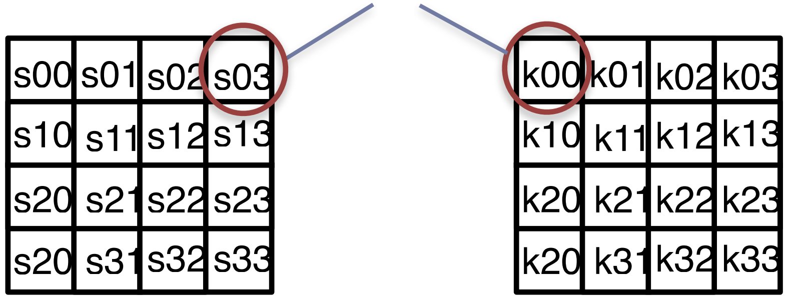


Last Round

Looking Inside AES Once Again

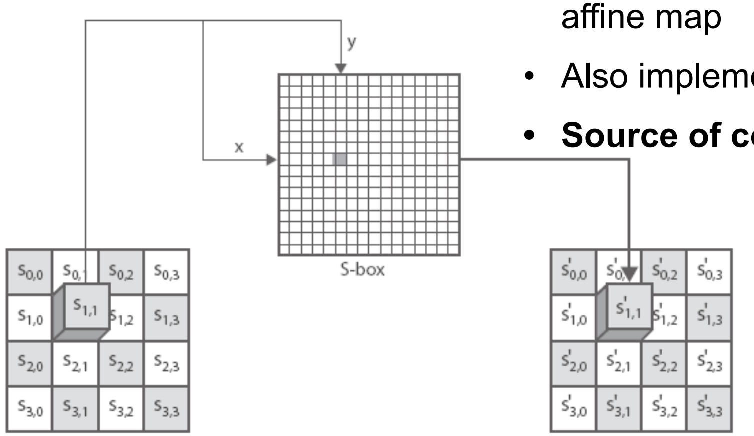
State

1 byte



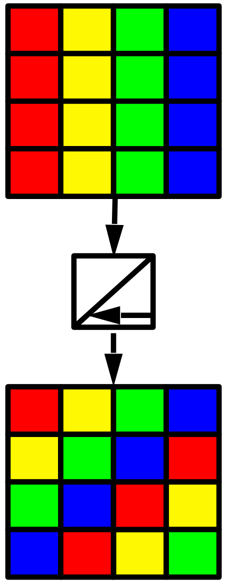
1. SubBytes

- Nonlinear Boolean Function
- Finite field inversion followed by affine map
- Also implemented as a table
- **Source of confusion**

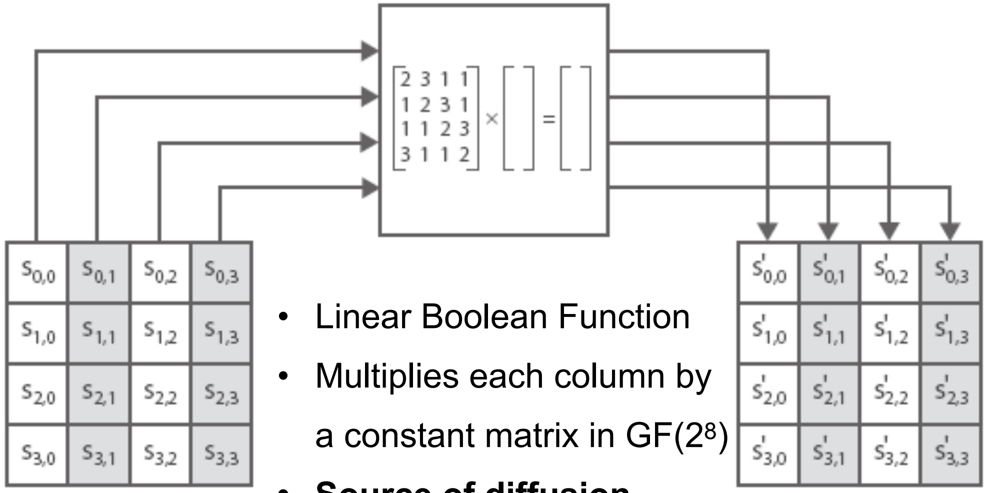


2. ShiftRows

- Linear Boolean Function
- Left circular shift of rows
- **Source of diffusion**

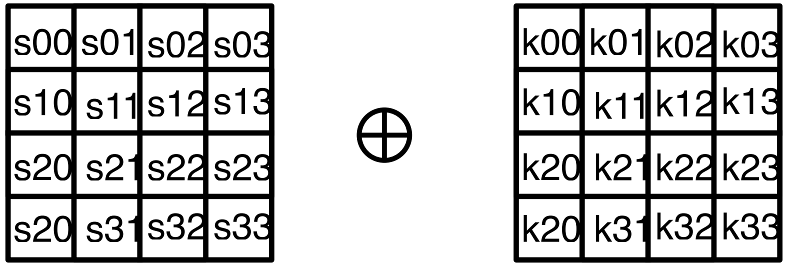


3. MixColumns



- Linear Boolean Function
- Multiplies each column by a constant matrix in $GF(2^8)$
- **Source of diffusion**

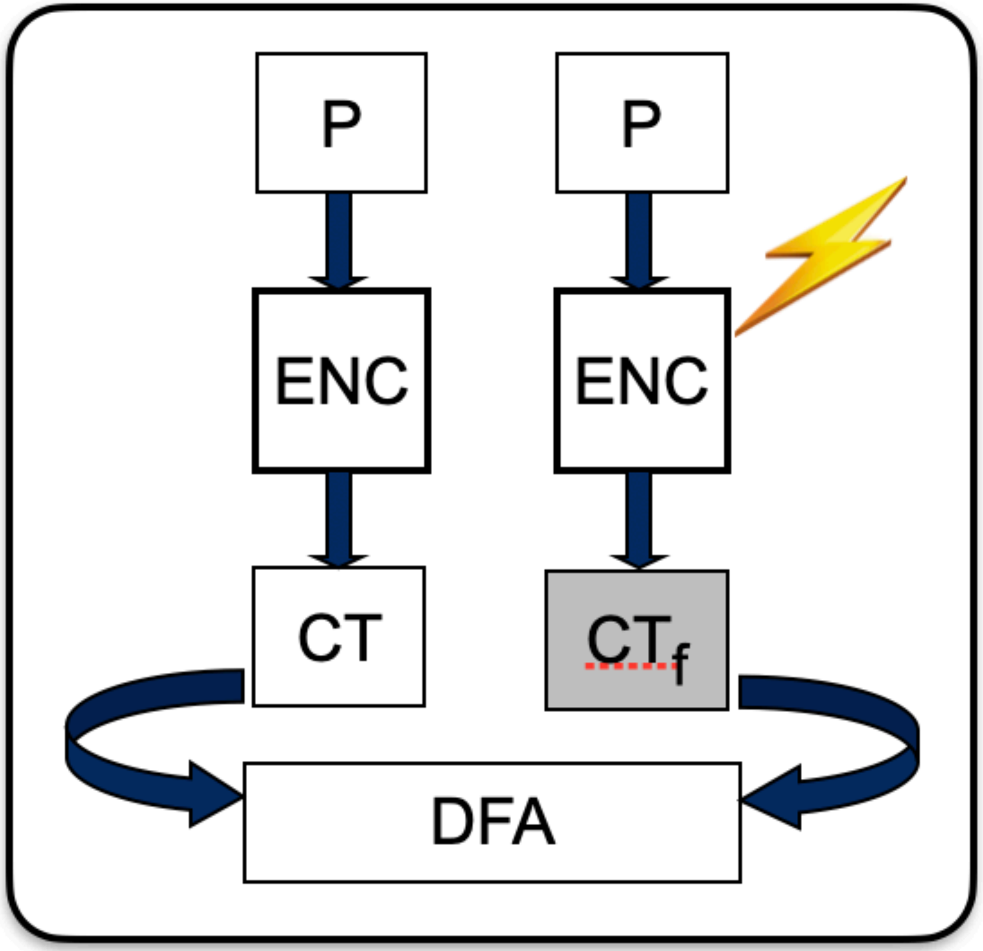
4. AddRoundKey



- Linear Boolean Function
- XOR the state with a round key

The AES Story…

- The attacker can corrupt one specific round operation of AES.
- One of multiple bytes of the AES state gets corrupted.
- Then what happens?
- We also assume that we have both correct and faulty ciphertext for the same plaintext…
- **Where to inject the fault**:
  - In round functions
  - In key schedule



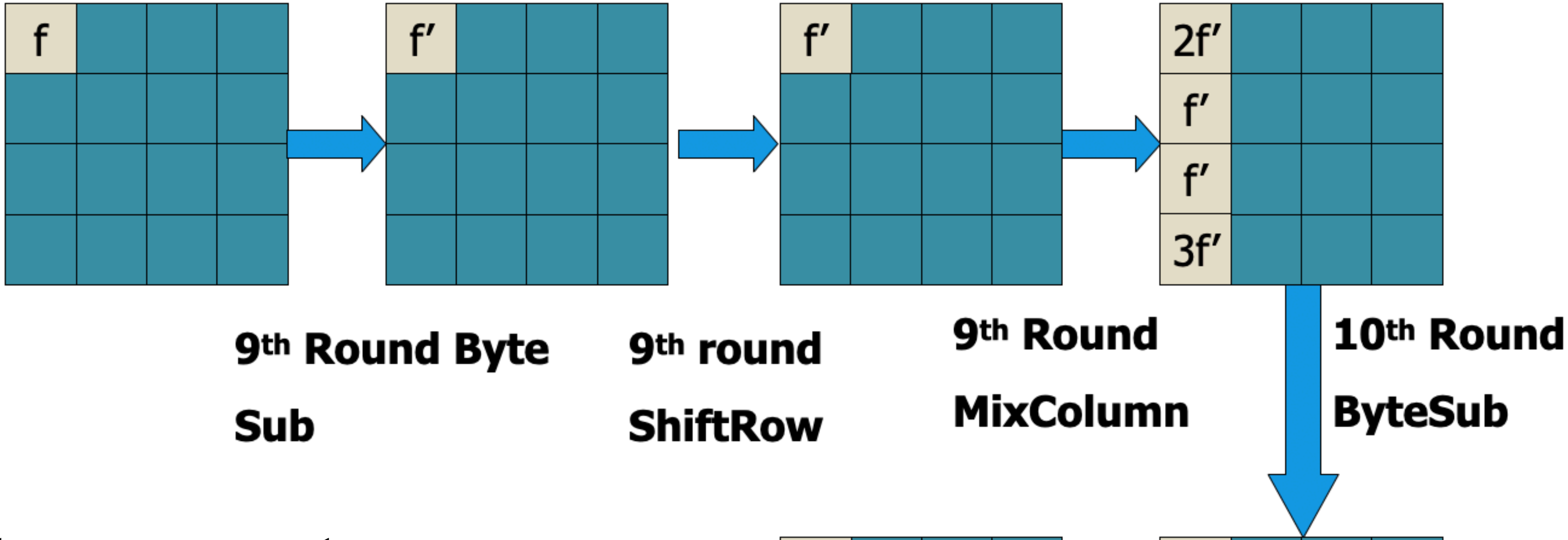
Let there be a fault at 9th round

Let the correct ciphertext be

$x = (x_1, x_2, \cdots x_{16})$

The faulty ciphertext be

$x' = (x'_1, x'_2, \cdots x'_{16})$

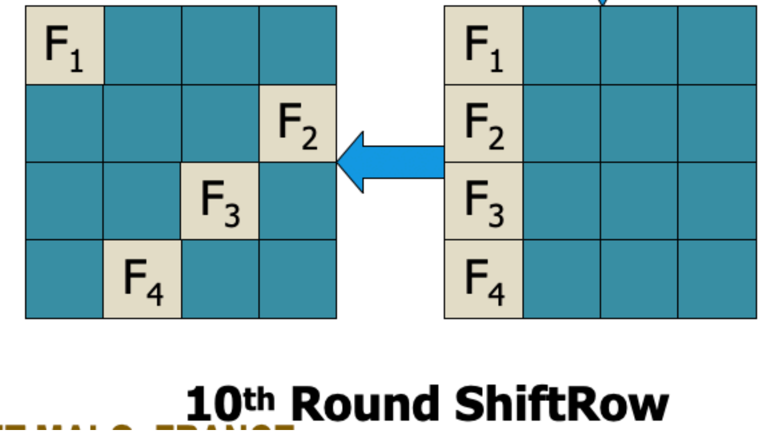


$$2f' = S^{-1}(x_1 \oplus k_1) \oplus S^{-1}(x'_1 \oplus k_1)$$

$$f' = S^{-1}(x_8 \oplus k_8) \oplus S^{-1}(x'_8 \oplus k_8)$$

$$f' = S^{-1}(x_{11} \oplus k_{11}) \oplus S^{-1}(x'_{11} \oplus k_{11})$$

$$3f' = S^{-1}(x_{14} \oplus k_{14}) \oplus S^{-1}(x'_{14} \oplus k_{14})$$



Let there be a fault at 9th round

$$2f' = S^{-1}(x_1 \oplus k_1) \oplus S^{-1}(x_1' \oplus k_1)$$

$$f' = S^{-1}(x_8 \oplus k_8) \oplus S^{-1}(x_8' \oplus k_8)$$

$$f' = S^{-1}(x_{11} \oplus k_{11}) \oplus S^{-1}(x_{11}' \oplus k_{11})$$

$$3f' = S^{-1}(x_{14} \oplus k_{14}) \oplus S^{-1}(x_{14}' \oplus k_{14})$$

- On an average there is one solution to the equation: $S^{-1}(X) \oplus S^{-1}(X + \alpha) = \beta$ — why?
  - Two possible solutions can be $X = 0$ and $X = \alpha$. In that case $\beta = \alpha^{-1}$.
  - If not, then we can transform this to $\beta x^2 + \alpha\beta x + \alpha = 0$ — which (may) have 2 more solutions
  - If $X = 0$ or $X = \alpha$, then the equation can have two more solutions in $GF(2^n)$, depending on n is even or odd. Solutions have form $\{0, \alpha, e\alpha, e^2\alpha\}$, with e being a field ($GF(2^n)$) element. — depends on some deep finite field tricks…

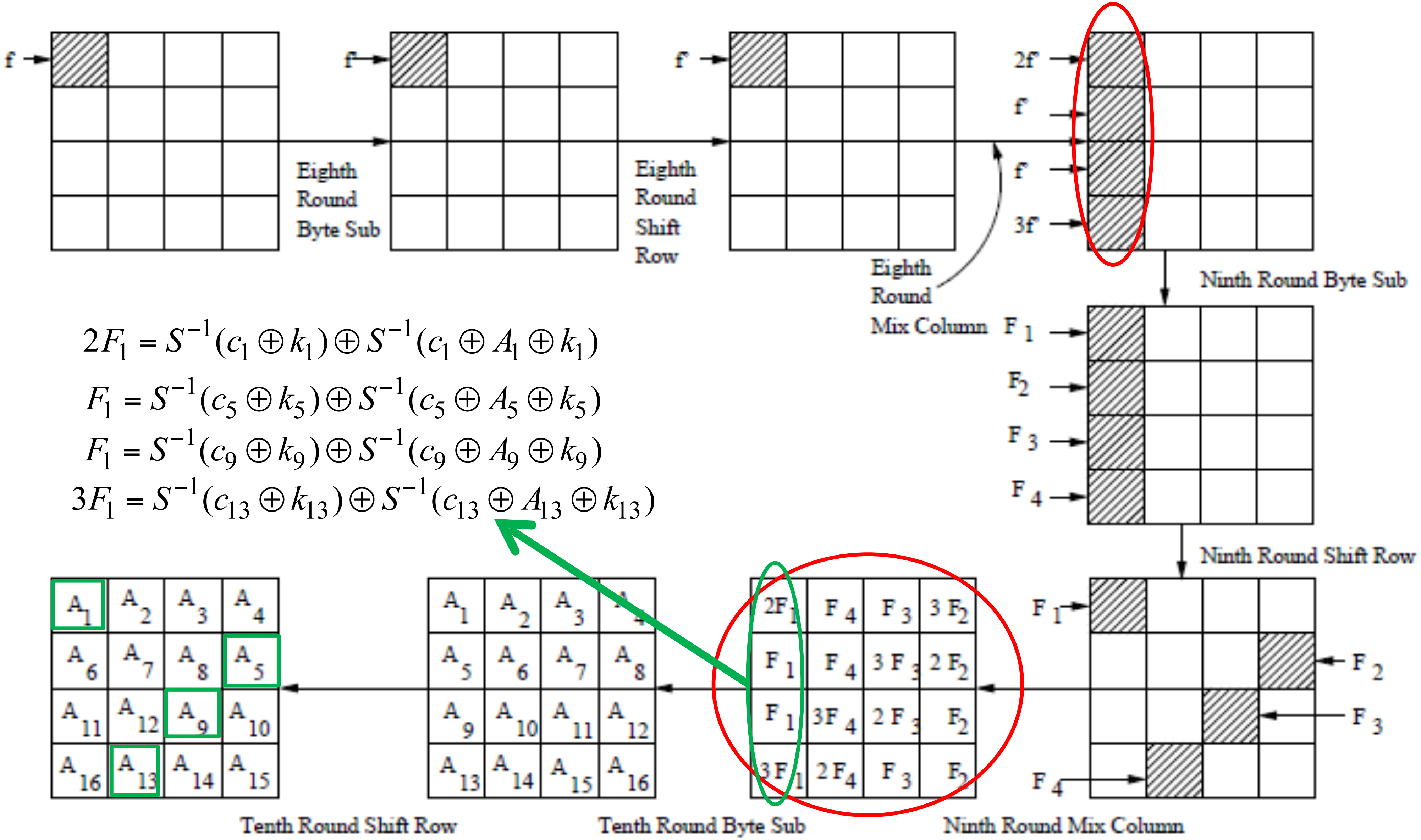
  So, it can have 0, 2, or 4 solutions — on average 1 solutions, we observed….
- Thus for one value of f′ there is 1 value for k_1, k_8, k_{11}, k_{14} which satisfies the equations.
- Thus for all the 2^8 values of f′, there are 2^8 values for k_1, k_8, k_{11}, k_{14}.
- Thus the total size of AES key is 2^{32}

Let there be a fault at 9th round

- With one faulty cipher text:
  - Number of possible values per 4 bytes of the key is around 2^8.
  - There are 2^{32} possible candidates for 128 bits of the AES key.
  - Brute force key is thus possible!

$$2f' = S^{-1}(x_1 \oplus k_1) \oplus S^{-1}(x_1' \oplus k_1)$$
$$f' = S^{-1}(x_8 \oplus k_8) \oplus S^{-1}(x_8' \oplus k_8)$$
$$f' = S^{-1}(x_{11} \oplus k_{11}) \oplus S^{-1}(x_{11}' \oplus k_{11})$$
$$3f' = S^{-1}(x_{14} \oplus k_{14}) \oplus S^{-1}(x_{14}' \oplus k_{14})$$

Let's Improve…8th Round Injection

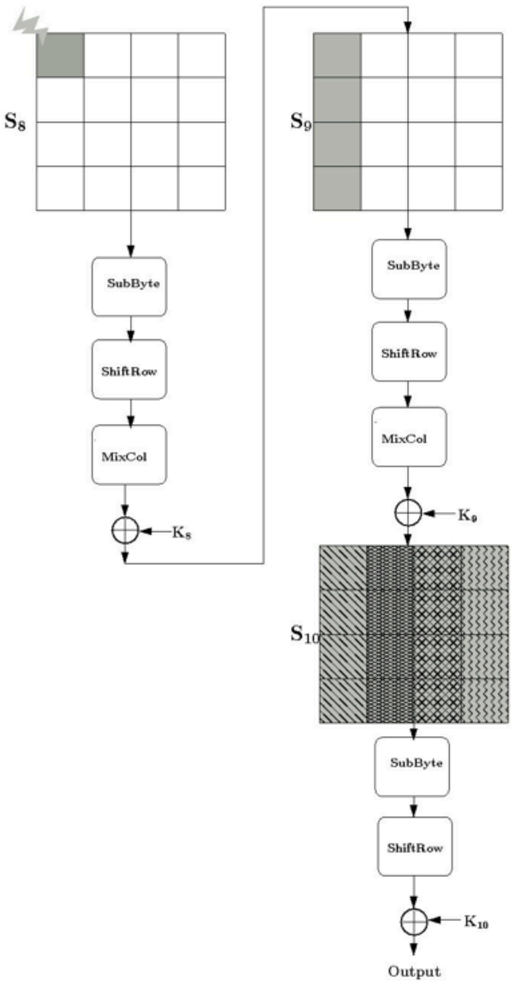


$$2F_1 = S^{-1}(c_1 \oplus k_1) \oplus S^{-1}(c_1 \oplus A_1 \oplus k_1)$$
$$F_1 = S^{-1}(c_5 \oplus k_5) \oplus S^{-1}(c_5 \oplus A_5 \oplus k_5)$$
$$F_1 = S^{-1}(c_9 \oplus k_9) \oplus S^{-1}(c_9 \oplus A_9 \oplus k_9)$$
$$3F_1 = S^{-1}(c_{13} \oplus k_{13}) \oplus S^{-1}(c_{13} \oplus A_{13} \oplus k_{13})$$

Let’s Improve…8th Round Injection…

- Search space reduced in two phases.
- First phase:
  - Find the 2^{32} candidates of 10^{th} round key.
- Second phase
  - Deduce four differential equation from differences {2f’,f’,f’,3f’}.
  - Reduce the 2^{32} candidates to 2^{8} using the four differential equation.

Let's Improve…8th Round Injection…



2^{32}

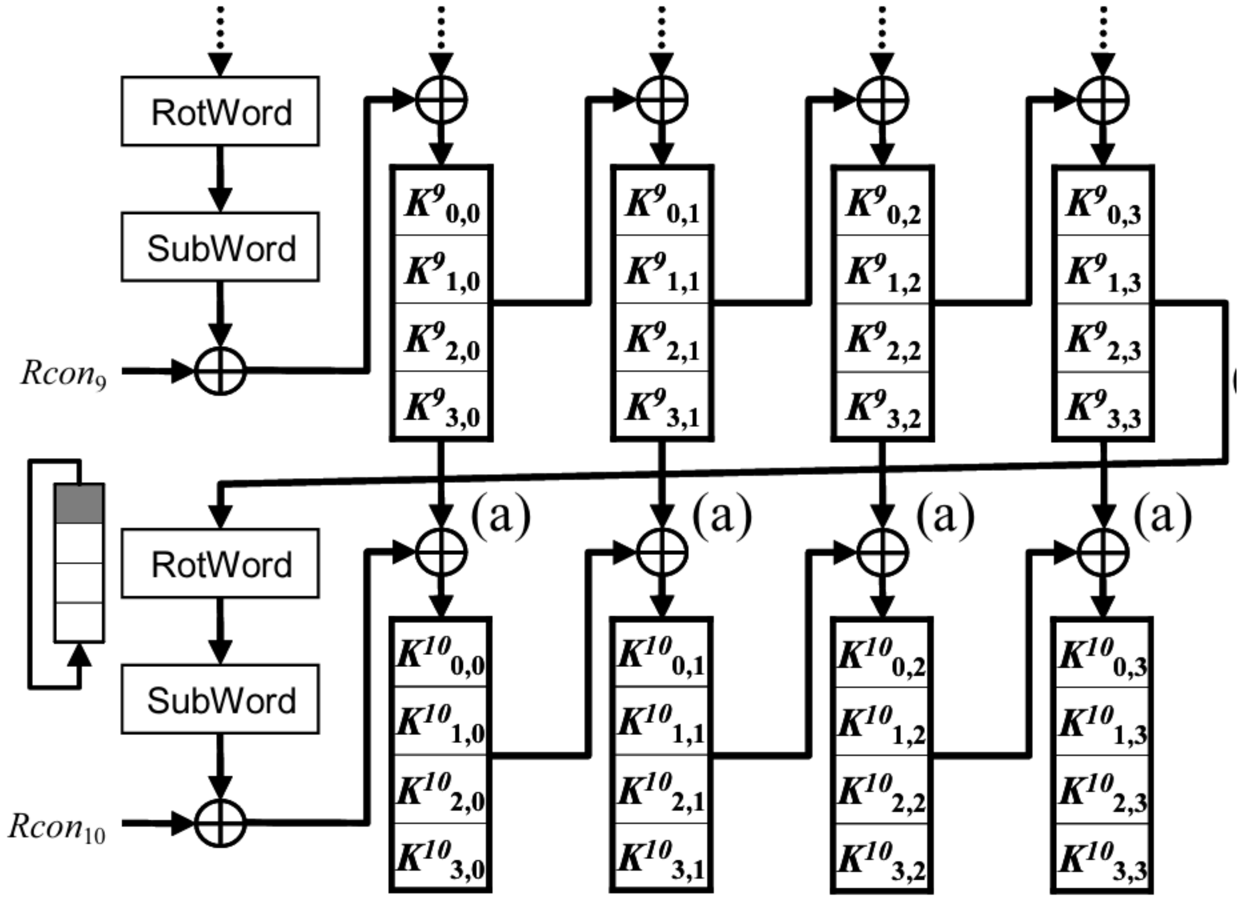
2^{128}

- Find 2^{32} candidates K_{10}

Differential Equation

$$2F_1 = S^{-1}(x_0 \oplus k_0) \oplus S^{-1}(x'_0 \oplus k_0)$$
$$F_1 = S^{-1}(x_{13} \oplus k_{13}) \oplus S^{-1}(x'_{13} \oplus k_{13})$$
$$F_1 = S^{-1}(x_{10} \oplus k_{10}) \oplus S^{-1}(x'_{10} \oplus k_{10})$$
$$3F_1 = S^{-1}(x_7 \oplus k_7) \oplus S^{-1}(x'_7 \oplus k_7)$$

AES Key Schedule



Rcon

Algorithm 2: The AES-128 `KeySchedule` function.

Input: $(r-1)^{th}$ round key ($X = x_i$ for $i \in \{1, \dots, 16\}$).
Output: r^{th} round key X.

for $i \leftarrow 0$ **to** 3 **do**
  $x_{(i<<2)+1} \leftarrow x_{(i<<2)+1} \oplus S(x_{(((i+1)\wedge 3)<<2)+4})$
end
$x_1 \leftarrow x_1 \oplus h_r$
for $i \leftarrow 1$ **to** 16 **do**
  if $(i-1) \mod 4 \neq 0$ **then**
    $x_i \leftarrow x_i \oplus x_{i-1}$
  end
end

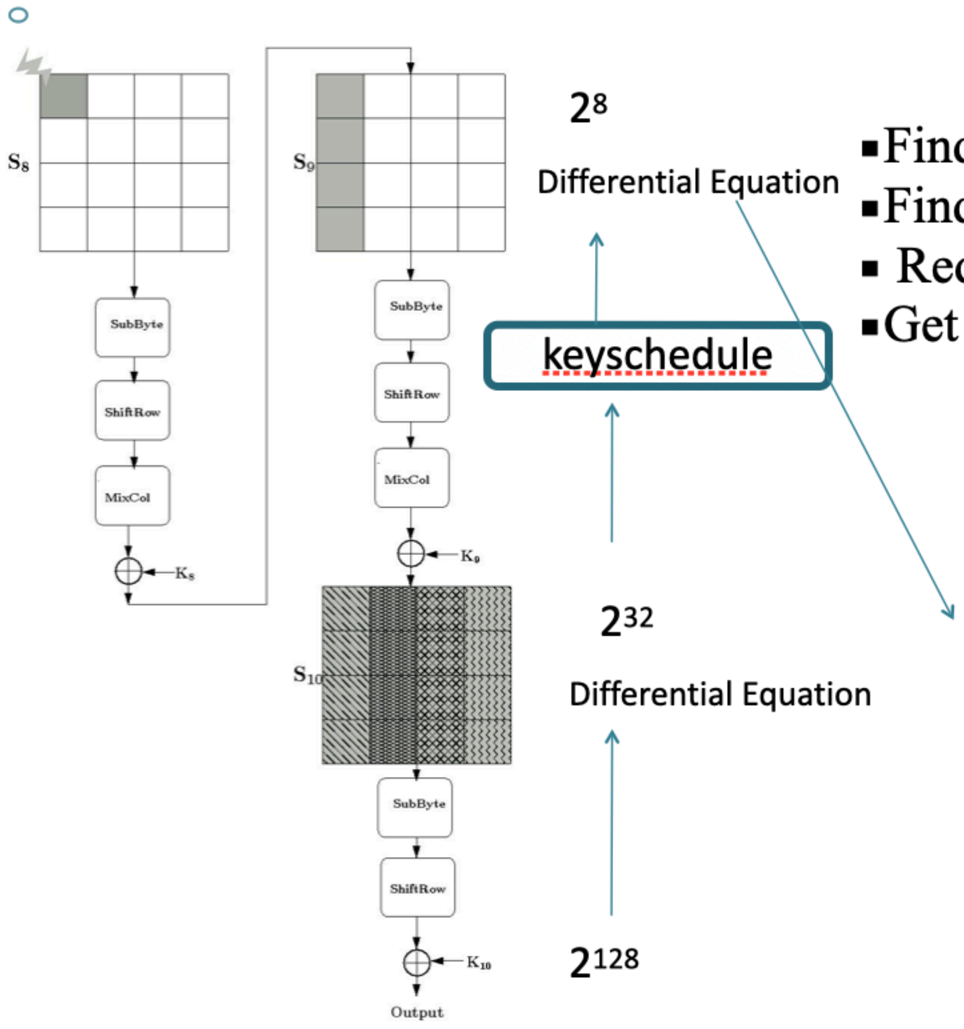
return X

Let's Improve…8th Round Injection…

$$\mathbf{K}^9 = \begin{pmatrix} k_1 \oplus S(k_{14} \oplus k_{10}) \oplus h_{10} & k_5 \oplus k_1 & k_9 \oplus k_5 & k_{13} \oplus k_9 \\ k_2 \oplus S(k_{15} \oplus k_{11}) & k_6 \oplus k_2 & k_{10} \oplus k_6 & k_{14} \oplus k_{10} \\ k_3 \oplus S(k_{16} \oplus k_{12}) & k_7 \oplus k_3 & k_{11} \oplus k_7 & k_{15} \oplus k_{11} \\ k_4 \oplus S(k_{13} \oplus k_9) & k_8 \oplus k_4 & k_{12} \oplus k_8 & k_{16} \oplus k_{12} \end{pmatrix} \cdot \quad \mathbf{K}^{10} = \{k_1, k_2, \cdots k_{16}\}$$

- AES key schedule is invertible

Let’s Improve…8th Round Injection…



- Find 2^{32} candidate K^{10}
- Find 2^{32} Candidates of K^9 using keyschedule
- Reduce K^9 to 2^8 candidates
- Get the master key by 2^8 brute-force search

$$2f' = \Delta S_9^{(0,0)}$$
$$f' = \Delta S_9^{(0,1)}$$
$$f' = \Delta S_9^{(0,2)}$$
$$3f' = \Delta S_9^{(0,3)}$$

8th Round Injection…How does the Equation Look?

$$
\begin{aligned}
2\,f' =\; & S^{-1}\Big(14\,\big(S^{-1}(x_1 \oplus k_1) \oplus k_1'\big) \oplus 11\big(S^{-1}(x_{14} \oplus k_{14}) \oplus k_2'\big) \oplus \\
& 13\,\big(S^{-1}(x_{11} \oplus k_{11}) \oplus k_3'\big) \oplus 9\,\big(S^{-1}(x_8 \oplus k_8) \oplus k_4'\big)\Big) \oplus \\
& S^{-1}\Big(14\,\big(S^{-1}(x_1' \oplus k_1) \oplus k_1'\big) \oplus 11\big(S^{-1}(x_{14}' \oplus k_{14}) \oplus k_2'\big) \oplus \\
& 13\,\big(S^{-1}(x_{11}' \oplus k_{11}) \oplus k_3'\big) \oplus 9\,\big(S^{-1}(x_8' \oplus k_8) \oplus k_4'\big)\Big) \\
=\; & S^{-1}\Big(14\,\big(S^{-1}(x_1 \oplus k_1) \oplus ((k_1 \oplus S(k_{14} \oplus k_{10}) \oplus h_{10}))\big) \oplus \\
& 11\,\big(S^{-1}(x_{14} \oplus k_{14}) \oplus (k_2 \oplus S(k_{15} \oplus k_{11}))\big) \oplus \\
& 13\,\big(S^{-1}(x_{11} \oplus k_{11}) \oplus (k_3 \oplus S(k_{16} \oplus k_{12}))\big) \oplus \\
& 9\,\big(S^{-1}(x_8 \oplus k_8) \oplus (k_4 \oplus S(k_{13} \oplus k_9))\big)\Big) \oplus \\
& S^{-1}\Big(14\,\big(S^{-1}(x_1' \oplus k_1) \oplus ((k_1 \oplus S(k_{14} \oplus k_{10}) \oplus h_{10}))\big) \oplus \\
& 11\,\big(S^{-1}(x_{14}' \oplus k_{14}) \oplus (k_2 \oplus S(k_{15} \oplus k_{11}))\big) \oplus \\
& 13\,\big(S^{-1}(x_{11}' \oplus k_{11}) \oplus (k_3 \oplus S(k_{16} \oplus k_{12}))\big) \oplus \\
& 9\,\big(S^{-1}(x_8' \oplus k_8) \oplus (k_4 \oplus S(k_{13} \oplus k_9))\big)\Big)
\end{aligned}
$$

- We have 4 such equations

Let’s Improve…8th Round Injection…

- Time complexity of previous attack: $O(2^{32})$
- Time complexity of this attack: $O(2^{8})$
- Just with one fault injection!!!

Diagonal Fault Attack

- Multi Byte Faults (more practical)
  - Attacker induces fault at the input of the 8^{th} round in some bytes
  - Fault value should be non-zero but can be arbitrary

Diagonal Fault Attack

b_{00}	b_{01}	b_{02}	b_{03}
b_{10}	b_{11}	b_{12}	b_{13}
b_{20}	b_{21}	b_{22}	b_{23}
b_{30}	b_{31}	b_{32}	b_{33}

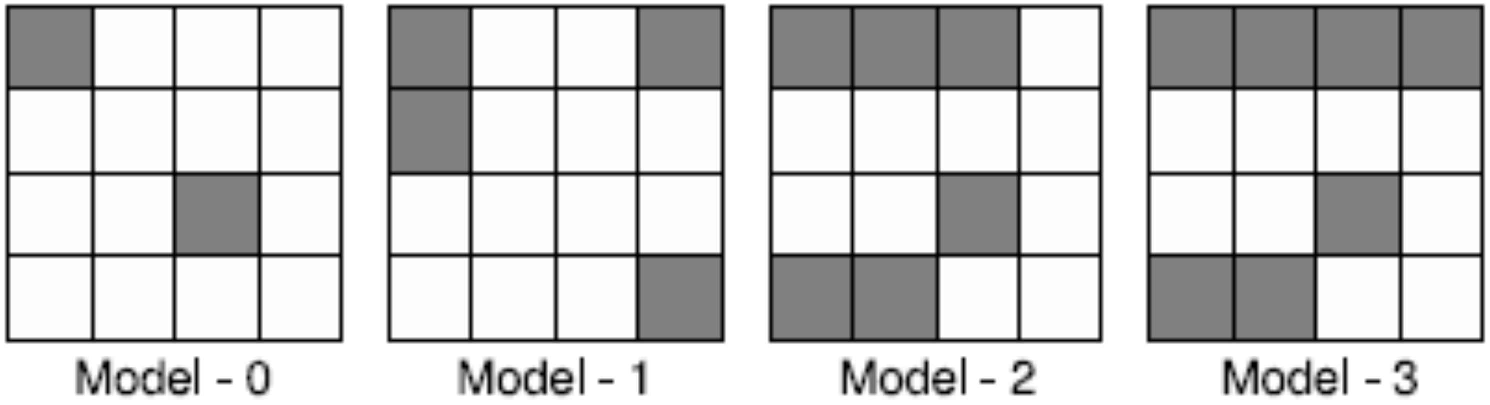
Diagonal: A diagonal is a set of four bytes of the state matrix,
where diagonal i is defined as follows:

$$D_i = \{b_{j,(j+i)mod4} \quad ; \quad 0 \leq j < 4\}$$

According to the above definition and with reference to the state matrix of AES (refer figure) we obtain the following four diagonals.

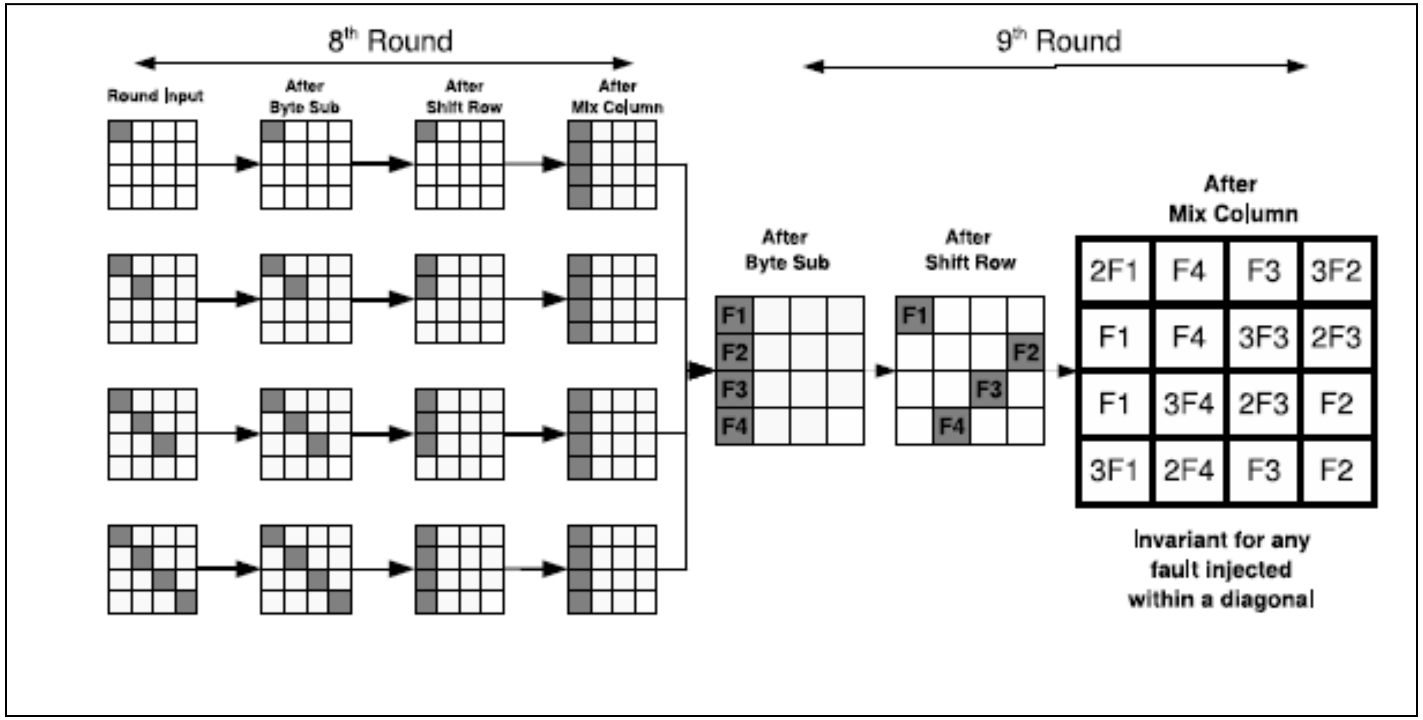
$$D_0 = (b_{00}, b_{11}, b_{22}, b_{33})$$
$$D_1 = (b_{01}, b_{12}, b_{23}, b_{30})$$
$$D_2 = (b_{02}, b_{13}, b_{20}, b_{31})$$
$$D_3 = (b_{03}, b_{10}, b_{21}, b_{32})$$

Diagonal Fault Attack



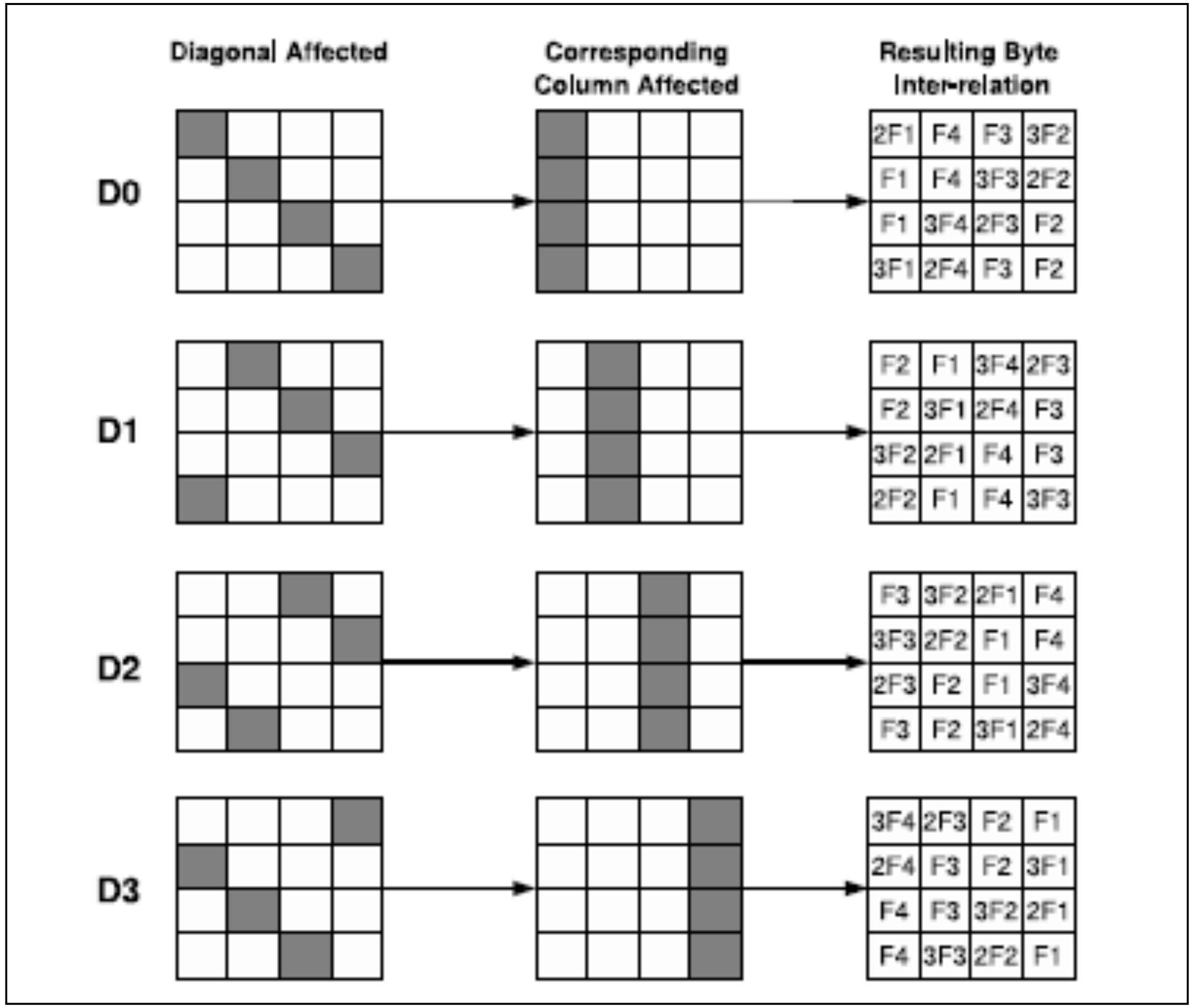
- M0: One Diagonal affected.
- M1: Two Diagonals affected.
- M2: Three Diagonals affected.
- M3: Four Diagonals affected.

Diagonal Fault Attack



- Faults induced in Diagonal D_0 at the input of 8th round AES are all equivalent.

Diagonal Fault Attack



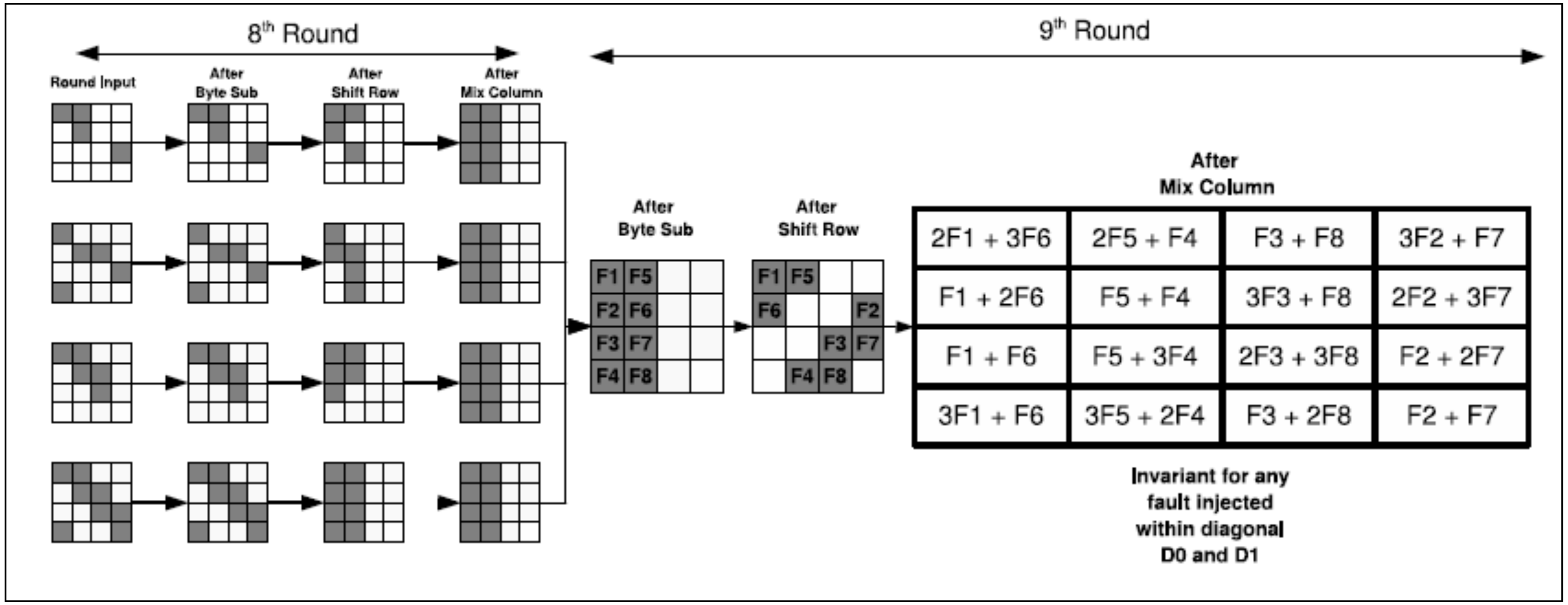
Diagonal Fault Attack

$$\mathbf{CT} = \begin{pmatrix} x_1 & x_2 & x_3 & x_4 \\ x_5 & x_6 & x_7 & x_8 \\ x_9 & x_{10} & x_{11} & x_{12} \\ x_{13} & x_{14} & x_{15} & x_{16} \end{pmatrix} \quad \mathbf{CT}' = \begin{pmatrix} x'_1 & x'_2 & x'_3 & x'_4 \\ x'_5 & x'_6 & x'_7 & x'_8 \\ x'_9 & x'_{10} & x'_{11} & x'_{12} \\ x'_{13} & x'_{14} & x'_{15} & x'_{16} \end{pmatrix} \quad \mathbf{K}_{10} = \begin{pmatrix} k_1 & k_2 & k_3 & k_4 \\ k_5 & k_6 & k_7 & k_8 \\ k_9 & k_{10} & k_{11} & k_{12} \\ k_{13} & k_{14} & k_{15} & k_{16} \end{pmatrix}$$

$$\boxed{\begin{aligned} ISB(x_1 + k_1) + ISB(x'_1 + k_1) &= 2[ISB(x_8 + k_8) + ISB(x'_8 + k_8)] \\ ISB(x_8 + k_8) + ISB(x'_8 + k_8) &= ISB(x_{11} + k_{11}) + ISB(x'_{11} + k_{11}) \\ ISB(x_{14} + k_{14}) + ISB(x'_{14} + k_{14}) &= 3[ISB(x_8 + k_8) + ISB(x'_8 + k_8)] \end{aligned}}$$

- **There are in total 4 such systems of equations for a diagonal D_0.**
- **Each system of equation gives 2^8 keys on an average.**
- **AES key size gets reduced to 2^{32}.**

Diagonal Fault Attack: 2 Diagonals Corrupted



$$a_0 = 2F_1 + 3F_6$$
$$a_1 = F_1 + 2F_6$$
$$a_2 = F_1 + F_6$$
$$a_3 = 3F_1 + F_6$$

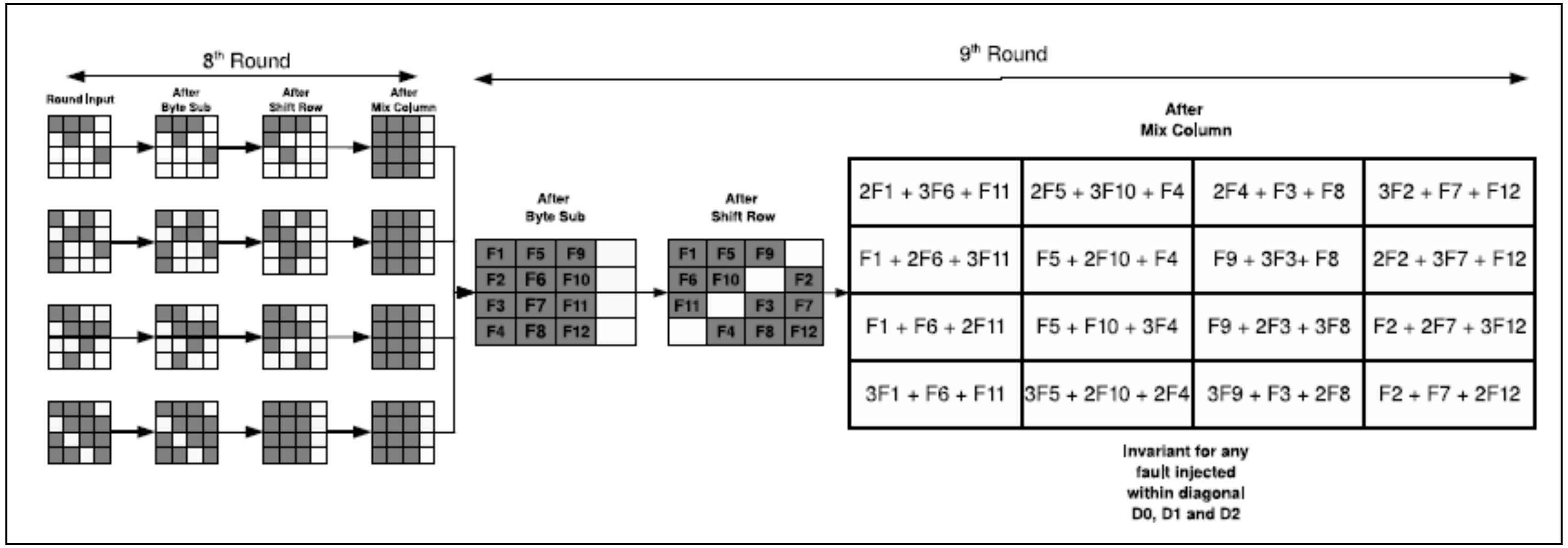
$$a_1 + a_3 = a_0$$
$$2a_1 + 3a_3 = 7a_2$$

Diagonal Fault Attack: 2 Diagonals Corrupted

$$\begin{aligned}
a_0 &= ISB(x_1 + k_1) + ISB(x'_1 + k_1) \\
a_1 &= ISB(x_8 + k_8) + ISB(x'_8 + k_8) \\
a_2 &= ISB(x_{11} + k_{11}) + ISB(x'_{11} + k_{11}) \\
a_3 &= ISB(x_{14} + k_{14}) + ISB(x'_{14} + k_{14})
\end{aligned}$$

- The equation reduces the space of the 4 key bytes of AES to 2^{16}
- Two faulty ciphertexts reduce it to a unique value on an average (experimental result).

Diagonal Fault Attack: 3 Diagonals Corrupted



$$
\begin{aligned}
a_0 &= 2F_1 + 3F_6 + F_{11} \\
a_1 &= F_1 + 2F_6 + 3F_{11} \\
a_2 &= F_1 + F_6 + 2F_{11} \\
a_3 &= 3F_1 + F_6 + F_{11}
\end{aligned}
$$

$$11a_0 + 13a_1 = 9a_2 + 14a_3$$

Diagonal Fault Attack: 3 Diagonals Corrupted

$$a_0 = ISB(x_1 + k_1) + ISB(x'_1 + k_1)$$
$$a_1 = ISB(x_8 + k_8) + ISB(x'_8 + k_8)$$
$$a_2 = ISB(x_{11} + k_{11}) + ISB(x'_{11} + k_{11})$$
$$a_3 = ISB(x_{14} + k_{14}) + ISB(x'_{14} + k_{14})$$

- The equation reduces the space of the 4 key bytes of AES to 2^{24}
- Four faulty ciphertexts reduce it to a unique value on an average (experimental result).

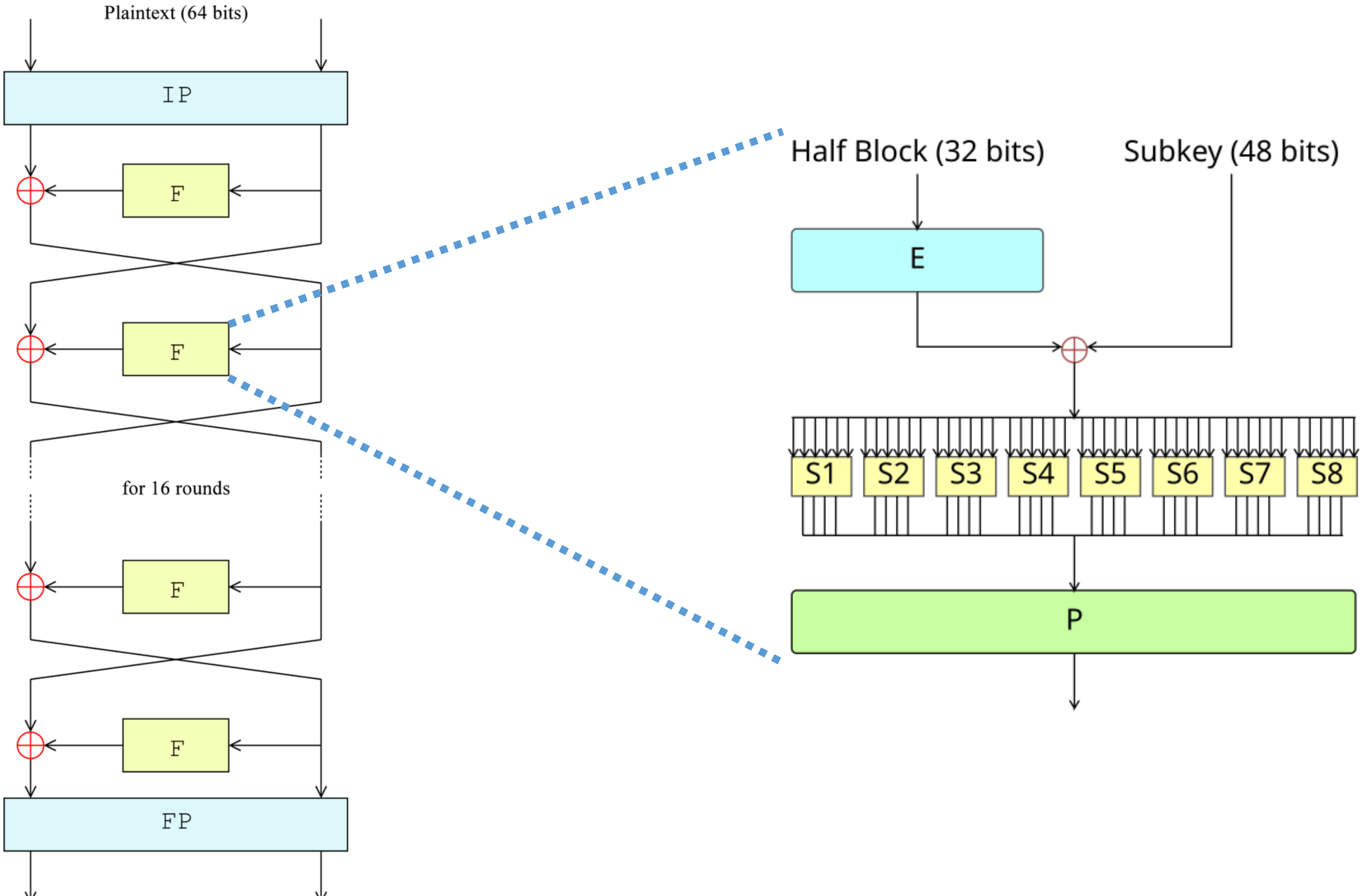
Diagonal Fault Attack: Practical Injection

Clock Frequency (MHz)	No Fault	Model 0 (M0)	Model 1 (M1)	Model 2 (M2)	Model 3 (M3)
36.0	512	0	0	0	0
36.1	512	0	0	0	0
36.2	512	0	0	0	0
36.3	510	2	0	0	0
36.4	511	1	0	0	0
36.5	508	4	0	0	0
36.6	504	8	0	0	0
36.7	507	5	0	0	0
36.8	490	22	0	0	0
36.9	489	23	0	0	0
37.0	419	79	14	0	0
37.1	448	60	4	0	0
37.2	434	64	13	1	0
37.3	403	94	15	0	0
37.4	408	99	5	0	0
37.5	248	226	38	0	0
37.6	214	205	84	9	0
37.7	128	205	122	57	0
37.8	76	180	133	123	0
37.9	20	122	145	225	0
38.0	158	191	129	34	0
38.1	27	116	185	185	0
38.2	40	127	198	147	0

37.9	20	122	145	225	0
38.0	158	191	129	34	0
38.1	27	116	185	185	0
38.2	40	127	198	147	0
38.3	26	69	155	257	5
38.4	17	62	137	254	42
38.5	0	20	68	361	63
38.6	0	0	16	319	177
38.7	0	2	20	293	197
38.8	0	1	8	290	213
38.9	0	11	42	368	91
39.0	15	59	107	308	23
39.1	0	2	12	197	301
39.2	0	5	26	339	142
39.3	0	3	11	285	213
39.4	0	0	0	134	378
39.5	0	0	6	138	368
39.6	0	0	0	150	362
39.7	0	0	0	21	491
39.8	0	0	0	18	494
39.9	0	0	0	14	498
40.0	0	0	0	0	512

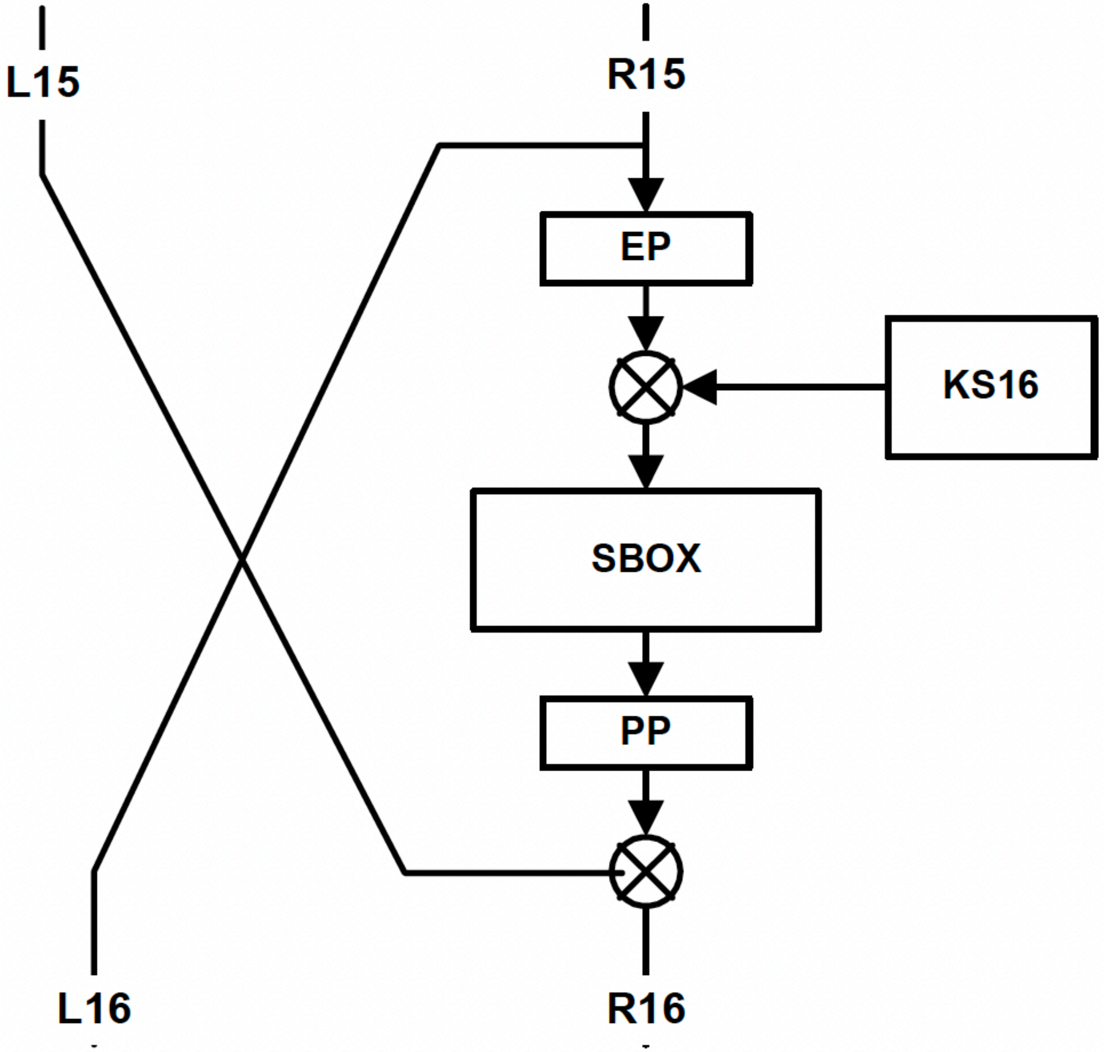
ATTACK REGION

Going Beyond AES...Let's Try DES

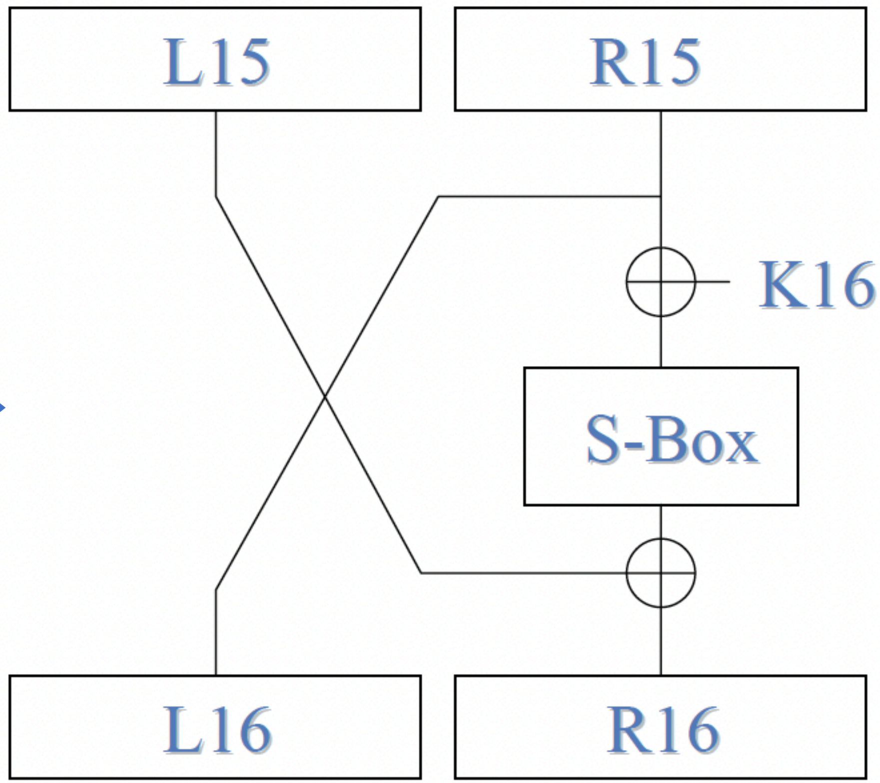


Source: Wikipedia

Going Beyond AES…Let's Try DES

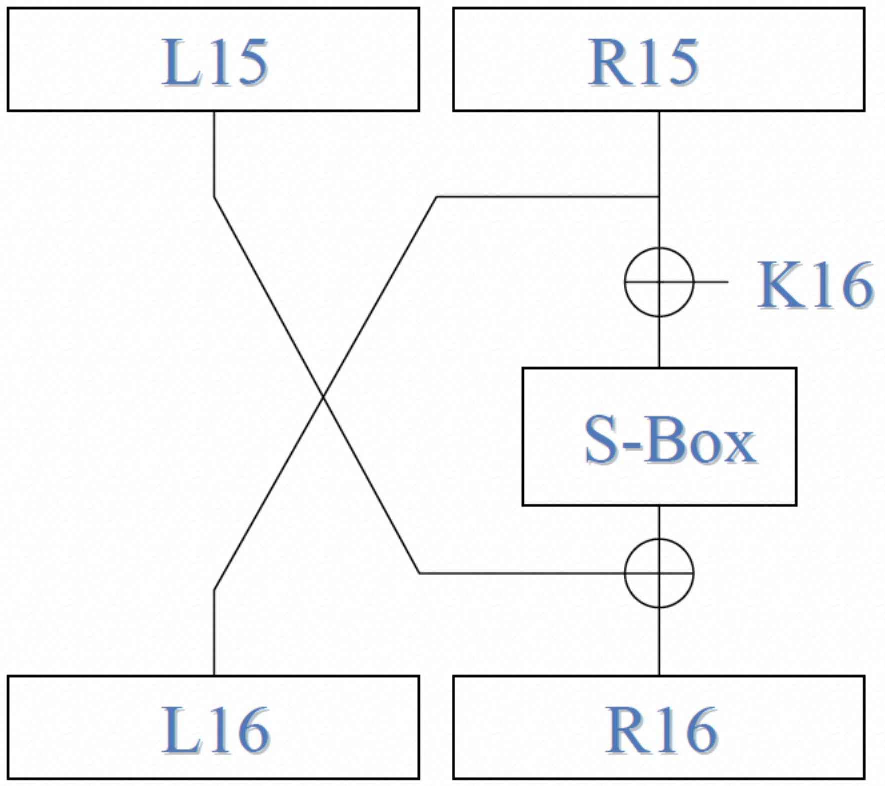


Simplify



- The Sorcerer's Apprenctice's Guide to Fault Attacks, FDTC 2006

Going Beyond AES…Let’s Try DES

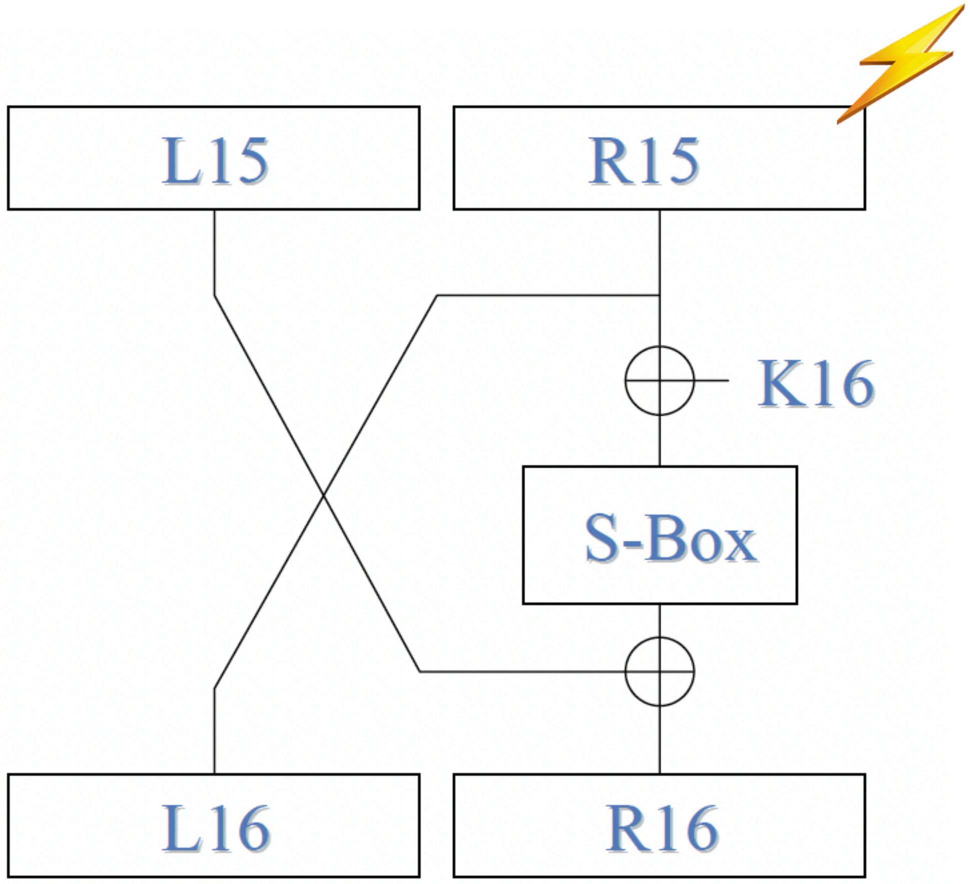


$$L16 = R15$$

$$R16 = S(R15 \oplus K16) \oplus L15$$

- The Sorcerer’s Apprenctice’s Guide to Fault Attacks, FDTC 2006

Going Beyond AES…Let's Try DES



$$L16 = R15$$

$$R16 = S(R15 \oplus K16) \oplus L15$$

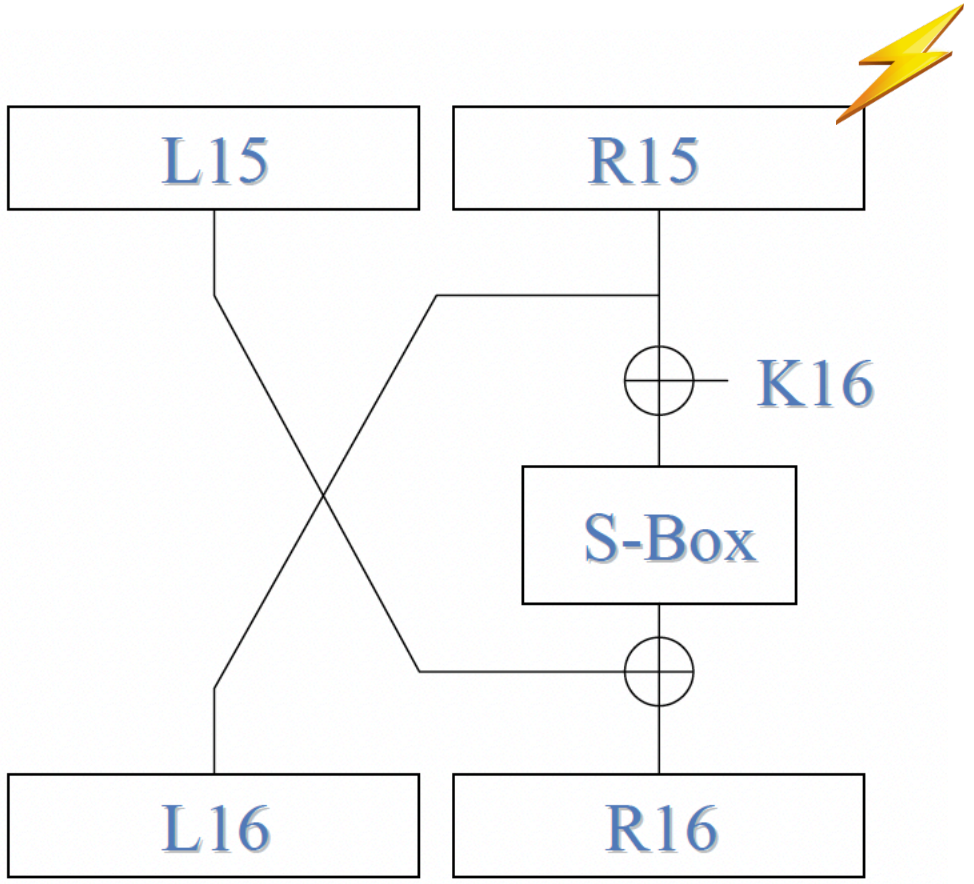
After Fault…

$$L16' = R15'$$

$$R16' = S(R15' \oplus K16) \oplus L15$$

- The Sorcerer's Apprenctice's Guide to Fault Attacks, FDTC 2006

Going Beyond AES…Let's Try DES



$$L16 = R15$$

$$R16 = S(R15 \oplus K16) \oplus L15$$

After Fault…

$$L16' = R15'$$

$$R16' = S(R15' \oplus K16) \oplus L15$$

Calculate the differential…

$$\begin{aligned} R16 \oplus R16' &= S(R15 \oplus K16) \oplus L15 \oplus S(R15' \oplus K16) \oplus L15 \\ &= S(L16 \oplus K16) \oplus S(L16' \oplus K16) \end{aligned}$$

- The Sorcerer's Apprenctice's Guide to Fault Attacks, FDTC 2006

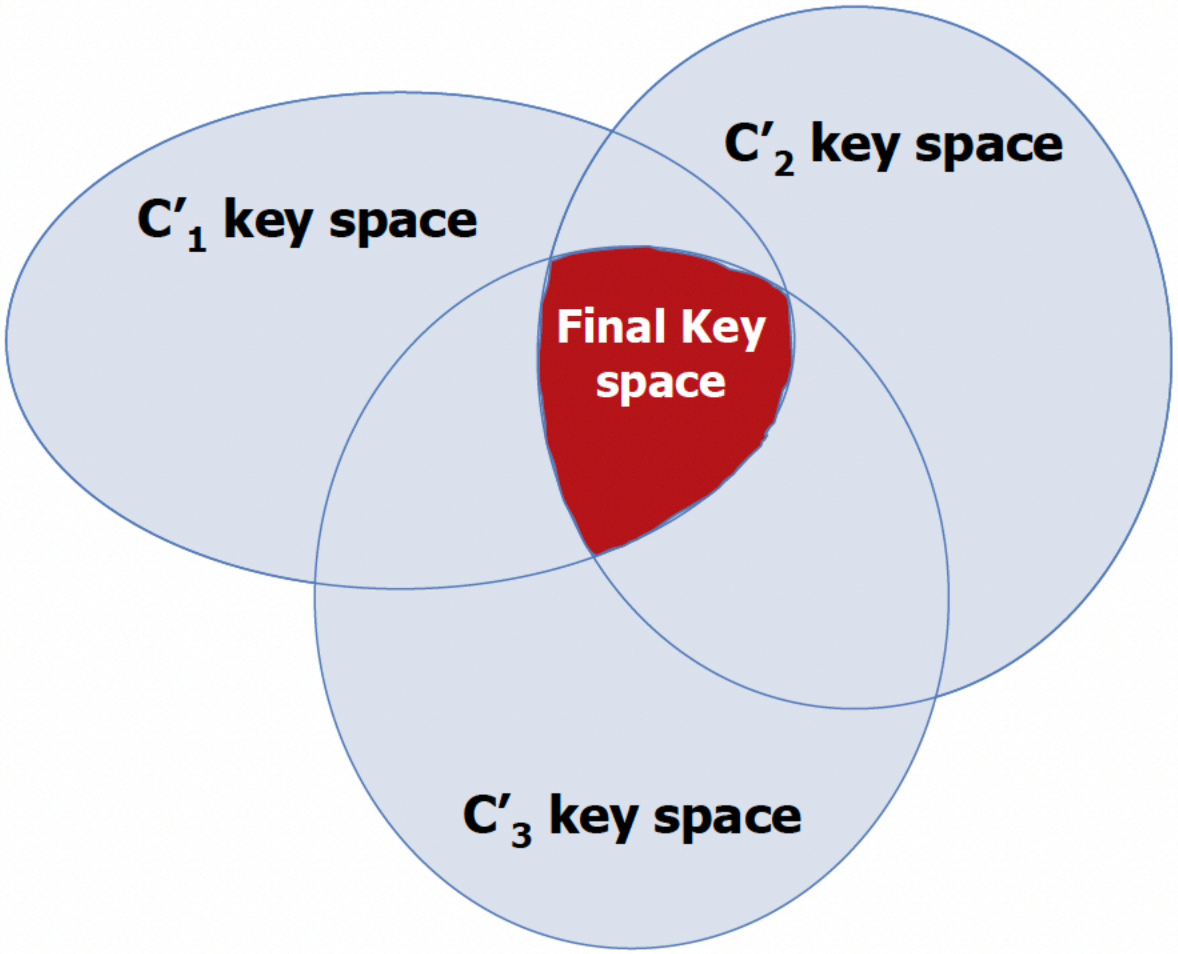
DES Attack

- Note that here both input and output differentials are known…
  - Unlike AES where only a pattern of the input differentials are known
- So, you can solve the equations for k as the only unknown
  - Note that DES S-Boxes are 6×4…so 6 bits has to be guessed simultaneously…
  - You can also use the differential distribution table mentioned in the last class to find the number of solutions for each S-Box

DES Attack

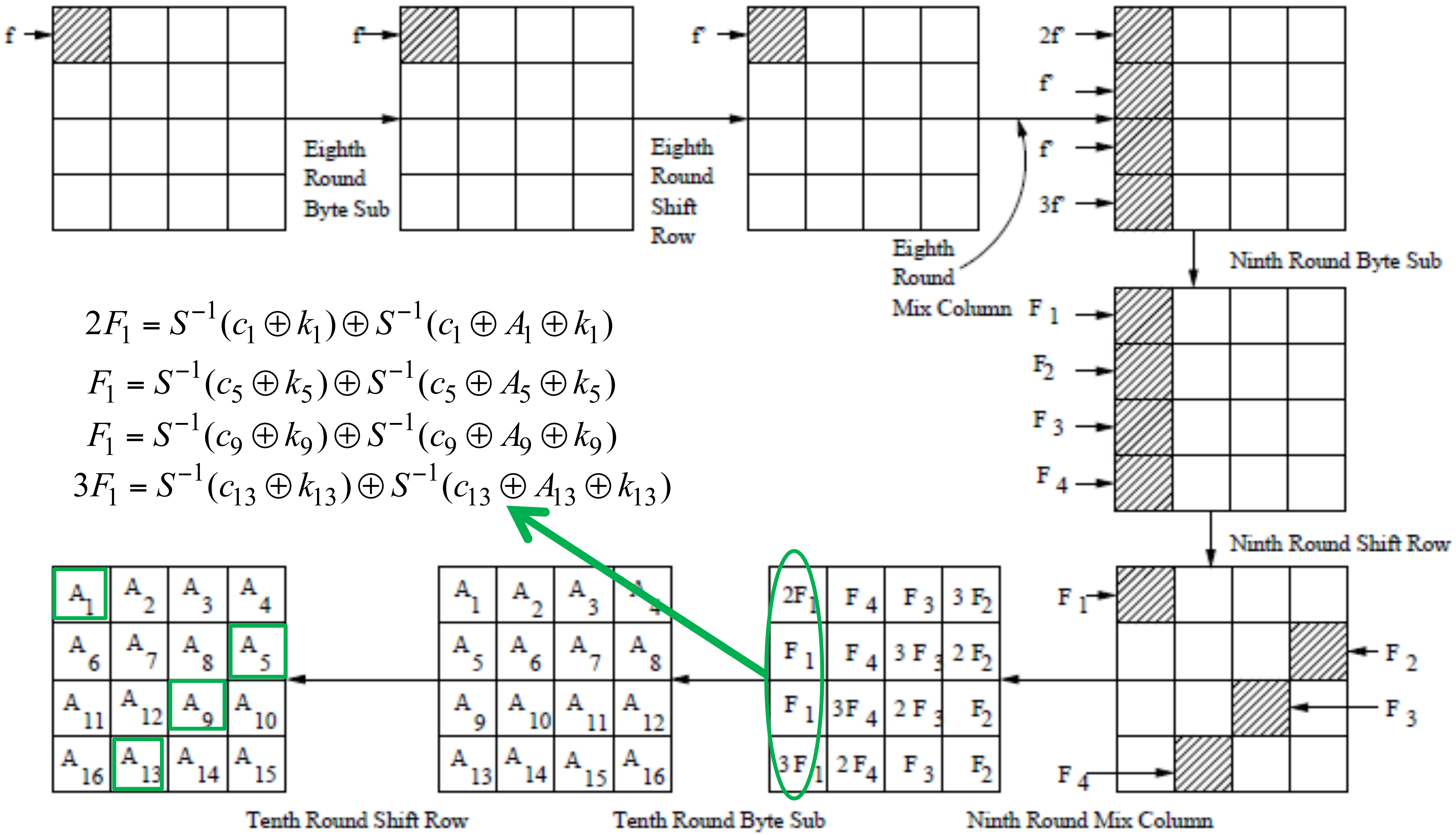
- Each equation gives 4-5 solutions for the key on average…
- So, for 8 S-boxs, roughly 2^{18} ($(5)^8$) suggestions remain for the last round key.
- Another issue with DES — although the key size is 56, round key size is 48.
- So 2^{48} reduces to 2^{18} — total master key can be found with $2^{18} \times 2^8 = 2^{26}$ exhaustive search…

Using Multiple Ciphertexts



- Common strategy for reducing attack complexity
  - Applies irrespective of AES or DES or any other cipher..
  - Let's try for AES…
  - In general, multiplying the number of faulty ciphertext directly reduces the final key space size and therefore the complexity of the exhaustive search

Let's Improve…8th Round Injection



$$2F_1 = S^{-1}(c_1 \oplus k_1) \oplus S^{-1}(c_1 \oplus A_1 \oplus k_1)$$
$$F_1 = S^{-1}(c_5 \oplus k_5) \oplus S^{-1}(c_5 \oplus A_5 \oplus k_5)$$
$$F_1 = S^{-1}(c_9 \oplus k_9) \oplus S^{-1}(c_9 \oplus A_9 \oplus k_9)$$
$$3F_1 = S^{-1}(c_{13} \oplus k_{13}) \oplus S^{-1}(c_{13} \oplus A_{13} \oplus k_{13})$$

Let's Improve…8th Round Injection

- For 1 column, a single injection reduces the key space from 2^{32} to 2^{8}.
- 2^{8} is basically the expected number of remaining subkeys
- This is done by satisfying the equation system.
- The probability of a given (4-byte) subkey satisfying this equation is $\frac{2^8}{2^{32}} = 2^{-24}$.
- Now, consider another fault injection
- We can take intersections between the key suggestions provided by these two injections…
- For 2-injections the probability will become $(2^{-24})^2$.
- Expected number of remain subkeys: $2^{32} \times (2^{-24})^2 < 1$ — **So practically only one subkey will remain**

$$
\begin{aligned}
2F_1 &= S^{-1}(c_1 \oplus k_1) \oplus S^{-1}(c_1 \oplus A_1 \oplus k_1) \\
F_1 &= S^{-1}(c_5 \oplus k_5) \oplus S^{-1}(c_5 \oplus A_5 \oplus k_5) \\
F_1 &= S^{-1}(c_9 \oplus k_9) \oplus S^{-1}(c_9 \oplus A_9 \oplus k_9) \\
3F_1 &= S^{-1}(c_{13} \oplus k_{13}) \oplus S^{-1}(c_{13} \oplus A_{13} \oplus k_{13})
\end{aligned}
$$

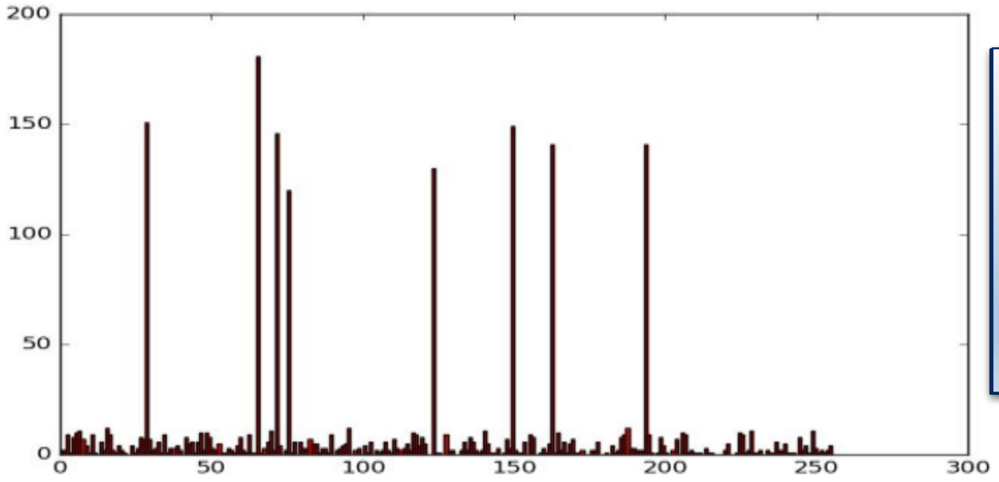
A General Model for Differential Fault Attacks

- **<u>Identify a distinguisher</u>**:
  - Based on fault patterns…
  - **Example**: For AES, 8th round injection, the distinguisher is the linear pattern observed due to fault propagation
  - For DES, the distinguisher is $R16 \oplus R16'$ — a known value and part of ciphertext!!!
  - Distiguisher is mostly an input differential Δ_i.
  - Always try to form equations of the form $S^{-1}(X) \oplus S^{-1}(X + \alpha) = \beta$, where β is a (part of) the distinguisher..
- **Divide-and Conquer**: Find a distinguisher evaluation strategy so that you do not need to guess a lot of keys together.
  - For AES attacks, this distinguisher evaluation complexity was 2^{36} — we have a divide-and conquer strategy which guesses 32-bits of key at once (independently for 4 columns)
  - For DES it is 2^9
- **Calculate the Remaining Key Space:** Estimate the number of solutions to your equation system. That gives you the number of remaining keys after the attack. You have to search these keys exhaustively…

Fault Attacks (FA)

Fault Models

- Key component of a fault attack
- Attack procedure changes according to the fault model
- Random localized faults
    - Bit/nibble/byte fault
    - Most general model
- **Biased faults**
    - **Device-dependent model**
- Instruction skip/modify.
- **Constant fault**
    - **Stuck-at-0/1**



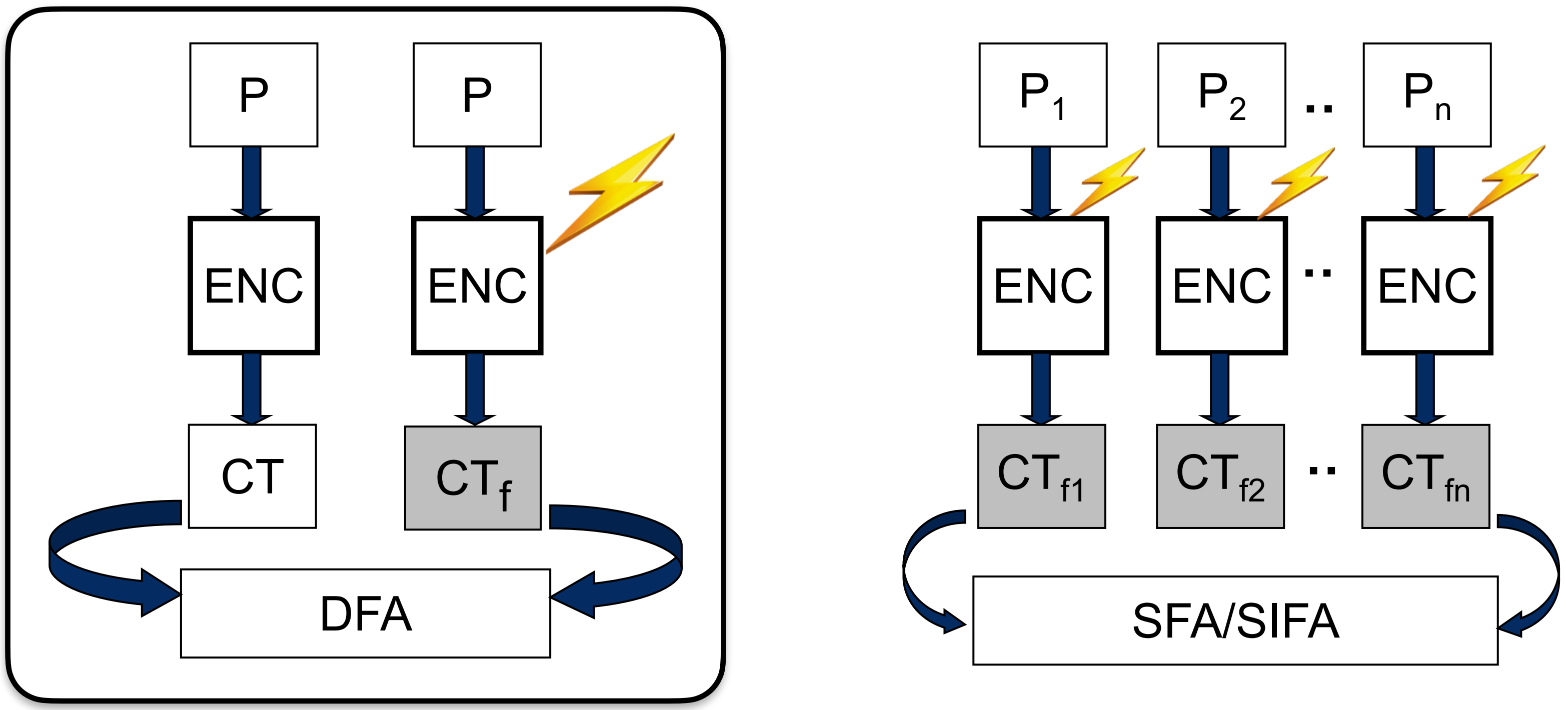
In a fault space of size 256, only 8 faults occur in 98% of the total injections!!!

```
ldi  r1  0;
ld   r1  #M1
ldi  r2  0
ld   r2  #M2
add  r1  r2
str  r1
```

```
ldi  r1  0;
nop
ldi  r2  0
ld   r2  #M2
add  r1  r2
str  r1
```

Fault Attacks (FA)

- Inject fault in the device during encryption/decryption.
- Analyze the faulty response to extract the secret
  - Differential fault attacks.
  - Biased fault attacks (SFA/SIFA).

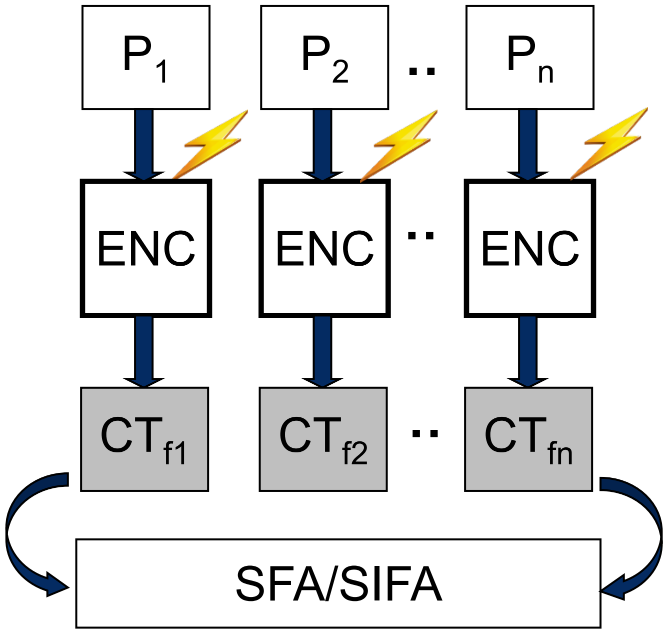


Statistical Fault Analysis (SFA)

<u>The Key Idea:</u>

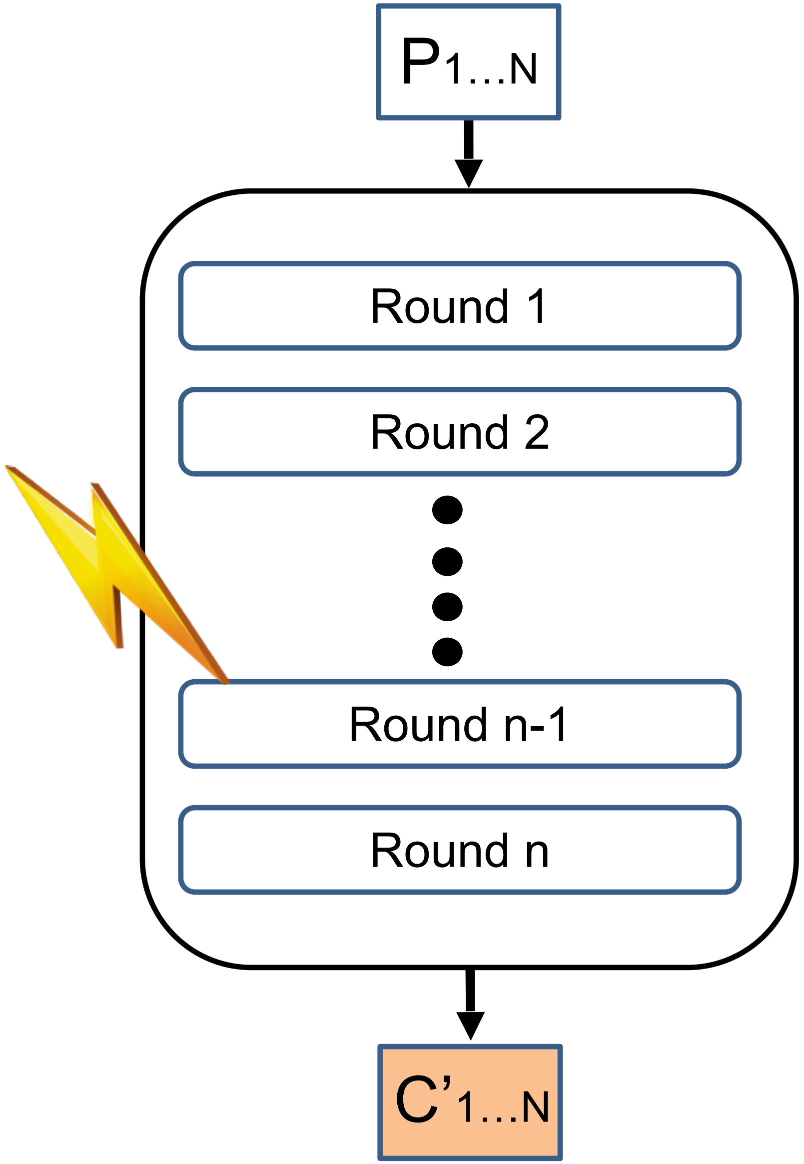
- Biased fault injection makes the state statistically biased.
- This bias is visible only for correct key guess.
- Such attacks require several faulty ciphertexts

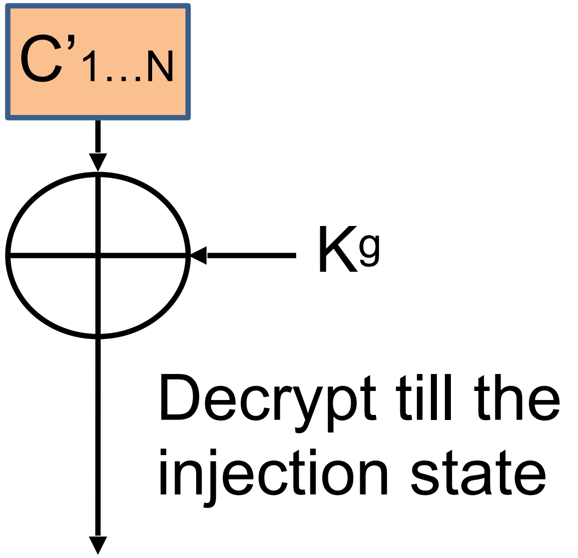
Correct State	Faulty State
0 0 0	0 0 0
0 0 1	0 0 0
0 1 0	0 1 0
0 1 1	0 1 0
1 0 0	1 0 0
1 0 1	1 0 0
1 1 0	1 1 0
1 1 1	1 1 0



<u>Example</u>: Consider a stuck-at-0 fault at the LSB bit of the intermediate state. The faulty state can only assume even values.

Statistical Fault Analysis (SFA)

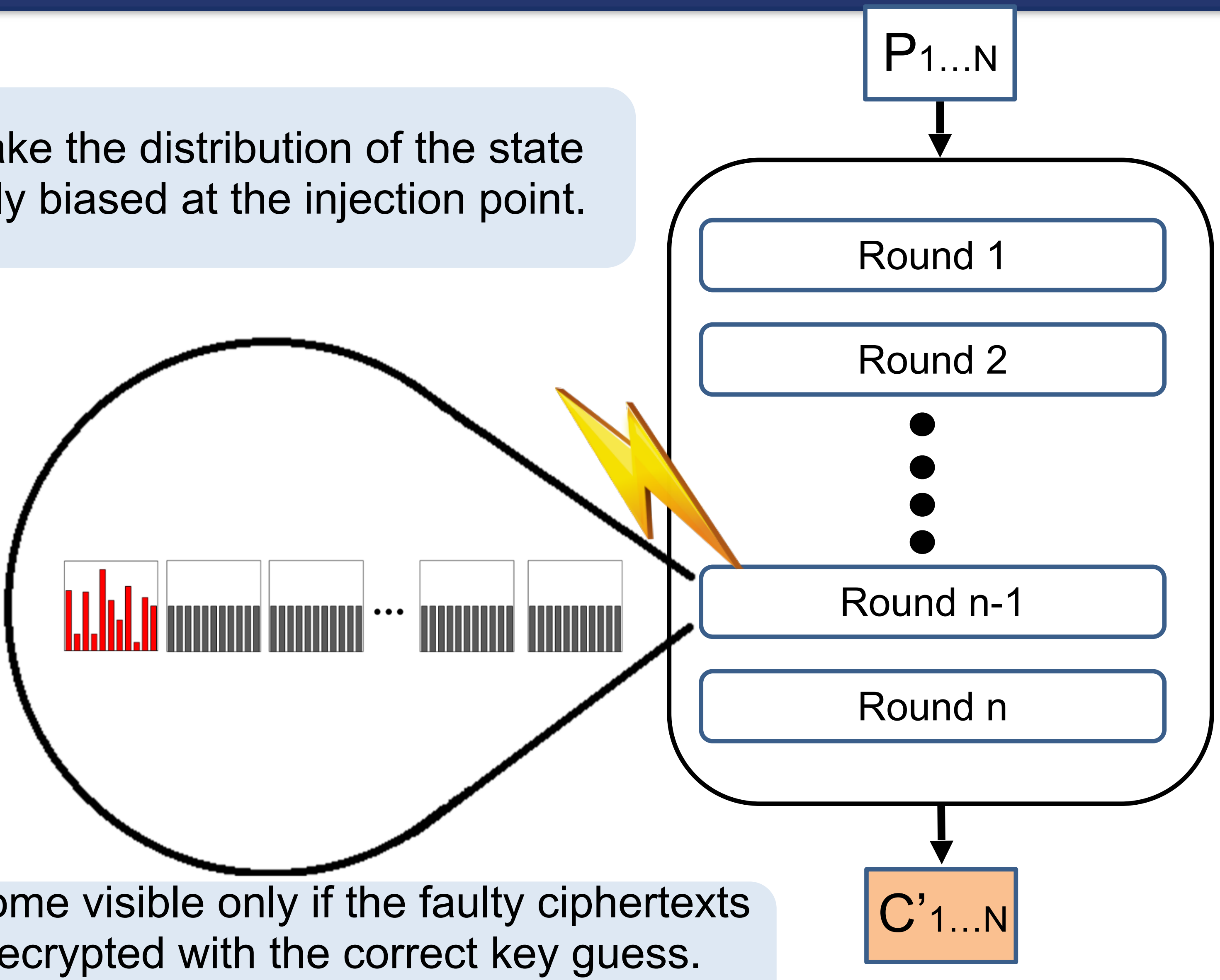




$$SEI(Kg) = \sum_{i=0}^{255}\left(\frac{\boxed{\#\{i \mid S^{-1}(C \oplus K_g) = i\}}}{n} - \frac{1}{256}\right)^2$$

Kg with highest SEI is the correct key

Faults make the distribution of the state statistically biased at the injection point.



The bias become visible only if the faulty ciphertexts are partially decrypted with the correct key guess. For wrong guesses the bias disappears.

Statistical Fault Analysis (SFA)

- Similar to SCA attacks in some sense…
- **Key reason**: The bias is only observable while the key guess is correct.
- **Interesting fact**: You do not need to know the exact bias
  - The SEI metric does not require exact statistical distribution of the bias.
- **Issues**: Can only be applied at the last few rounds. Not like DFA which can also be applied at the middle rounds.

Statistical Fault Analysis (SFA)

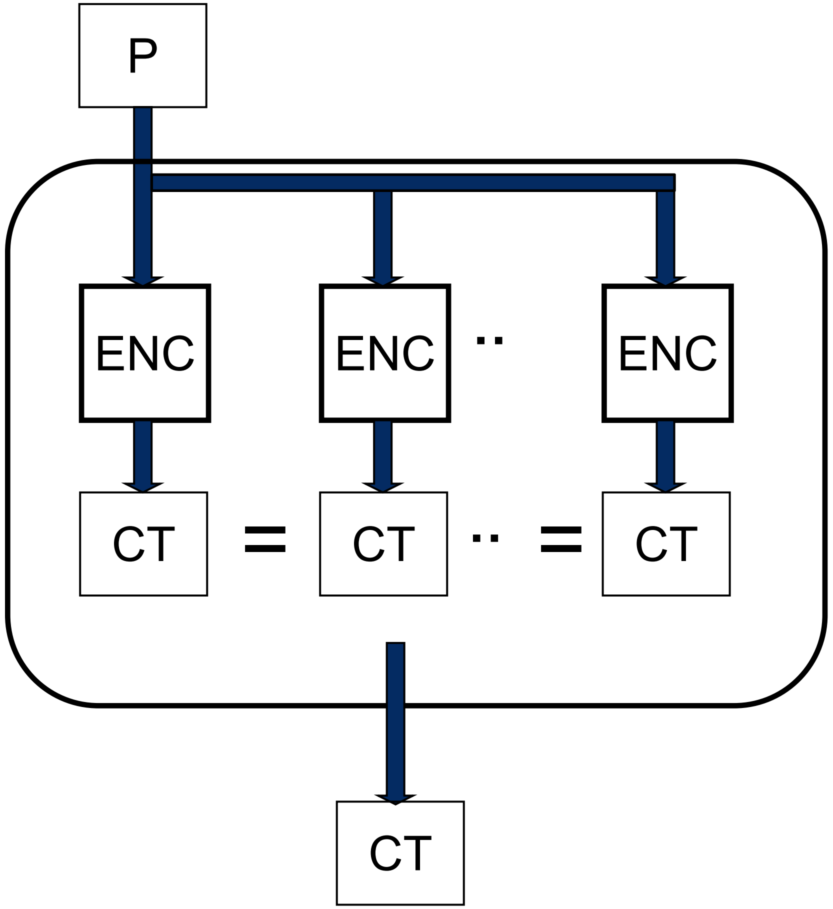
- Similar to SCA attacks in some sense…
- **Key reason**: The bias is only observable while the key guess is correct.
- **Interesting fact**: You do not need to know the exact bias
  - The SEI metric does not require exact statistical distribution of the bias.
- **Issues**: Can only be applied at the last few rounds. Not like DFA which can also be applied at the middle rounds.

Today

- Fault attack Countermeasures
- Breaking and fixing FA countermeasures…

It’s just Redundant Computation

- Do same computation several times.
- We call it redundant branches of computation.
- **When output/internal computation does not match it simply outputs a random value or outputs nothing…**
- Also called **concurrent error detection (CED)**
- **Assumption**: The adversary cannot fault all the branches with same valued fault.
- Motivated by ideas from classical fault tolerance literature.
- Error-detection/Correction codes.



What is an Error-Detection/Correction Code

- Adds some redundancy in data so a fixed number of erroneous bits can be detected/corrected.

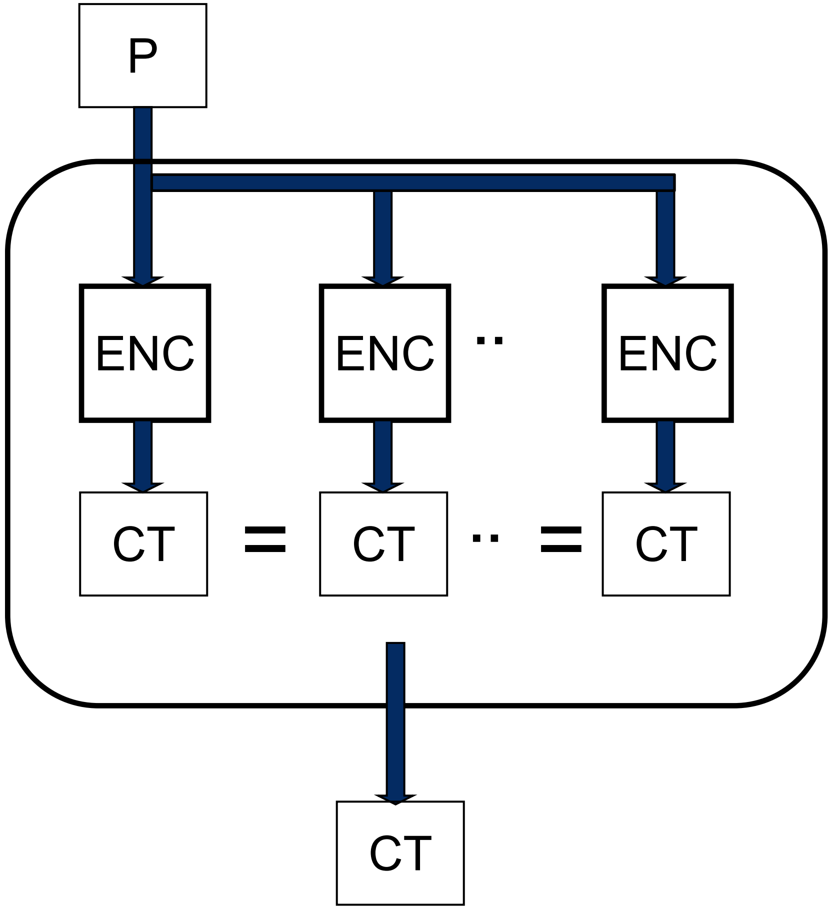
Definition 1 (Linear Code). *A binary linear $[n, k]$-code $\mathbf{C}$ with length n and rank k is defined as a vector subspace over $\mathbb{F}_2^n$ which maps messages $x \in \mathbb{F}_2^k$ to codewords $c \in \mathbf{C}$.*

Definition 2 (Generator Matrix). *A $k \times n$-matrix G is a generator matrix of an $[n, k]$-code $\mathbf{C}$ iff it consists of k basis vectors of $\mathbf{C}$ with length n. It can be used to map every message $x \in \mathbb{F}_2^k$ to its corresponding codeword with $x \cdot G = c \in \mathbf{C}$.*

Definition 3 (Minimum Distance). *The minimum distance d of a linear $[n, k, d]$-code $\mathbf{C}$ is defined as*

$$d = \min \left\{ wt\left(c_1 \oplus c_2\right) \mid c_1, c_2 \in \mathbf{C}, c_1 \neq c_2 \right\},$$

where $wt : \mathbb{F}_2^n \mapsto \mathbb{N}$ denotes the Hamming weight.



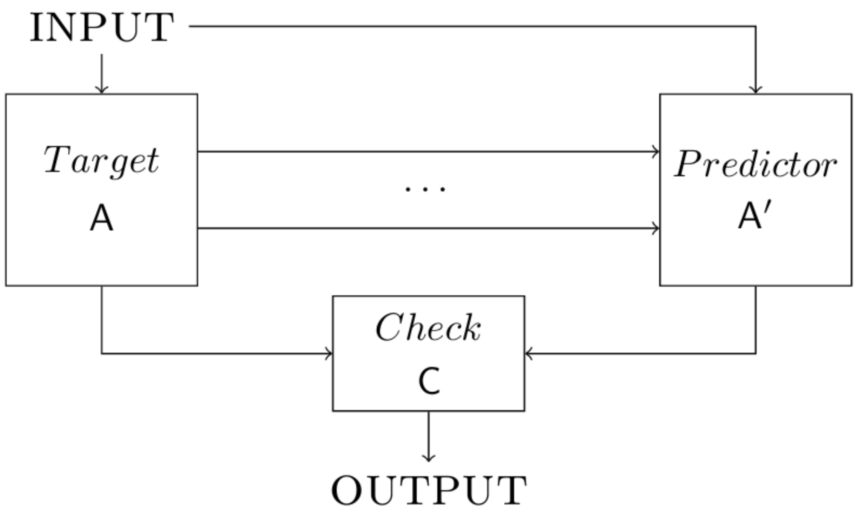
What is an Error-Detection/Correction Code

Lemma 1. *A linear $[n,k,d]$-code can detect all erroneous codewords $c' = c \oplus e$ with $wt(e) < d$.*

Lemma 2. *A linear $[n,k,d]$-code can correct all erroneous codewords $c' = c \oplus e$ with $wt(e) < d/2$.*

In other words, a linear $[n,k,d]$-code can detect errors up to $(d-1)$ bits, or correct errors up to $(d-1)/2$ bits.

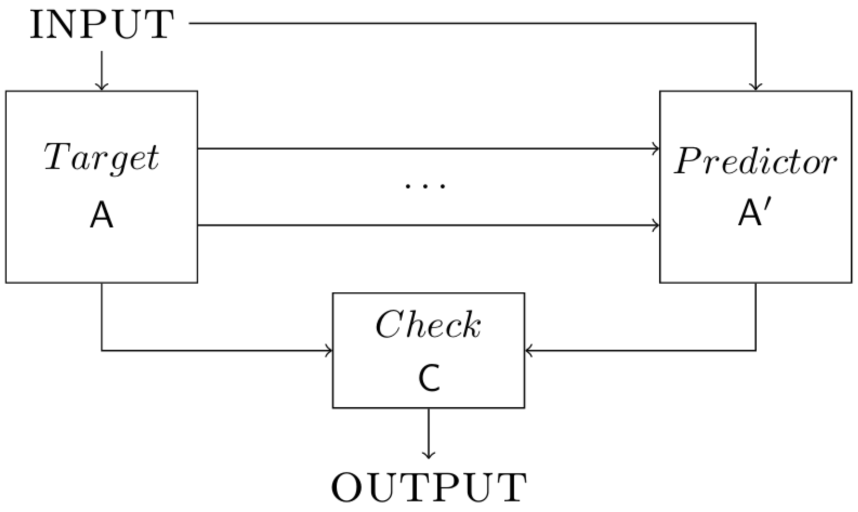
Definition 4 (Systematic Code). *The generator matrix G of a systematic code $\mathbf{C}$ is of the form $G = [I_k|P]$ where I_k denotes the identity matrix of size k.*



- What is the advantage of systematic code?
  - Each message x is padded with check bits x' where $x' = x \,.\, P$
  - $c = < x | x' >$
  - So, you can process the message and check bits separately.

What is an Error-Detection/Correction Code

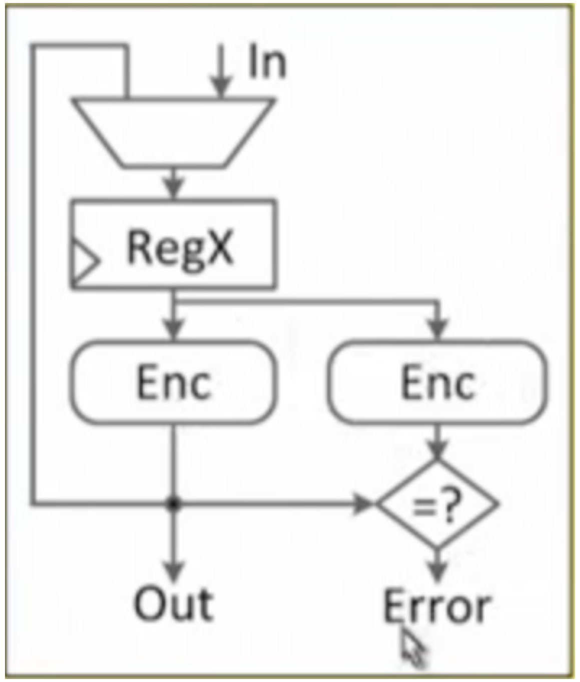
- Example 1:
  - Simply repeat your computation λ times.
  - It turns into a code $[\lambda k, k, \lambda]$.
  - Duplication: $\lambda = 2$
  - Triple modular redundancy: $\lambda = 3$
  - Overhead: λ times extra computation



- Example 2:
  - Add a parity bit as $x' = \bigoplus_k x$. — simply compute the XOR of all bits
  - $[k+1, k, 2]$ code, with error-detection capability 1 bits. — no correction
  - To check the received word y, $\bigoplus_k y = \bigoplus_k x$

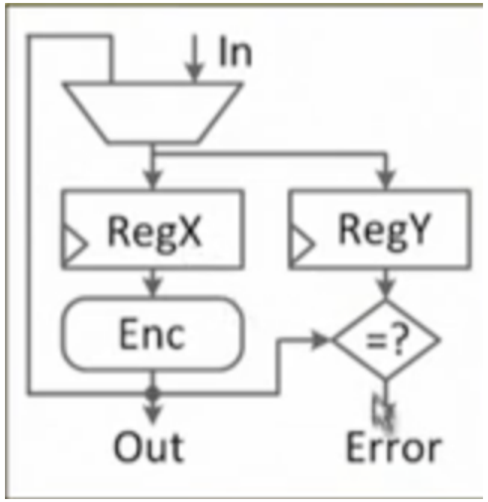
Challenges of CED: Hardware Vs. Time Tradeoff

- **Hardware Redundancy**:
  - Keep separate hardware for the predictor
  - Almost 100% overhead for simple duplication
  - Parity is sightly better, but for many other codes the overhead can be similar to duplication
  - **Generally, simple duplication is easier to deal with and allows several architectural choices as we see next**

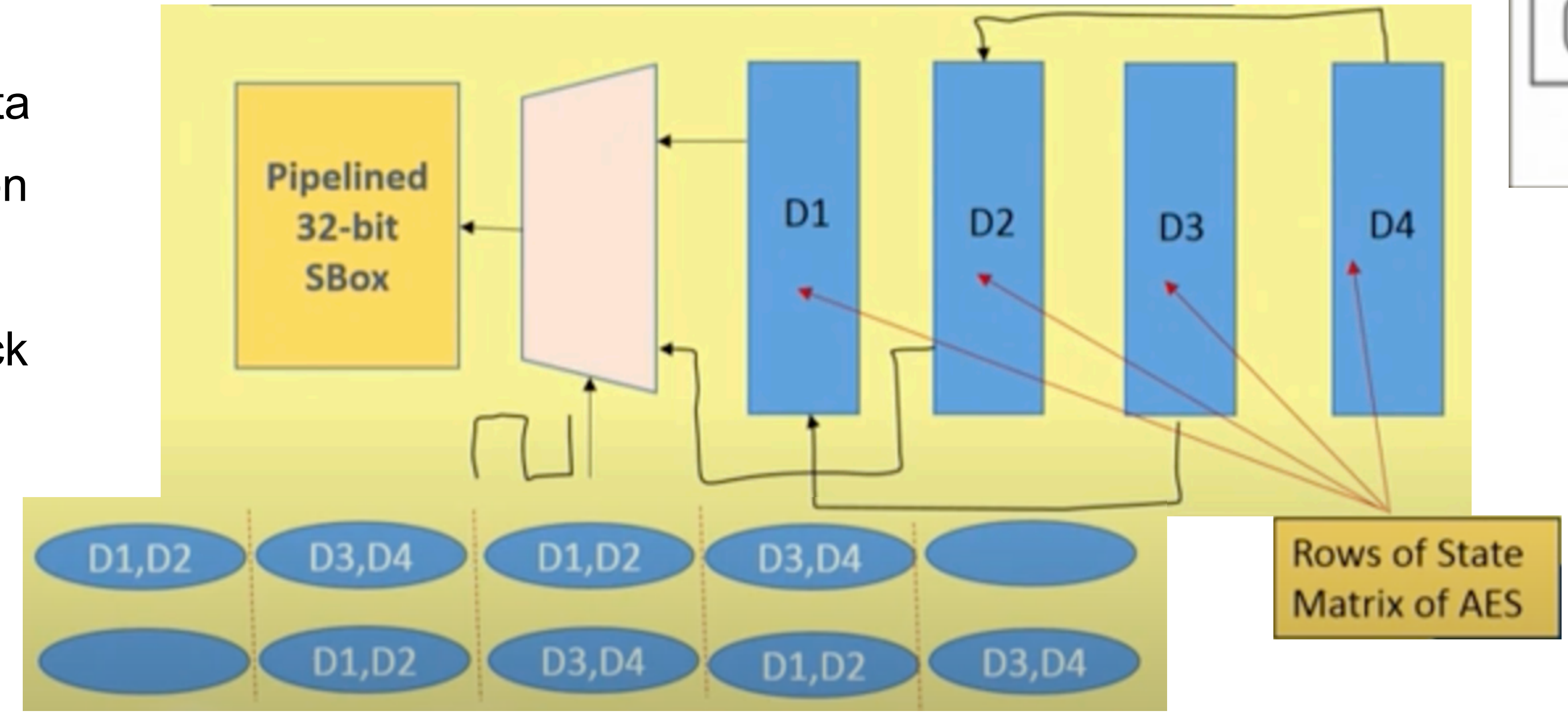


Challenges of CED: Hardware Vs. Time Tradeoff

- **Time Redundancy**:
  - Use the same hardware/code, but run it twice
  - Negligible hardware overhead, but throughput is low
  - Efficient design can hide some overhead



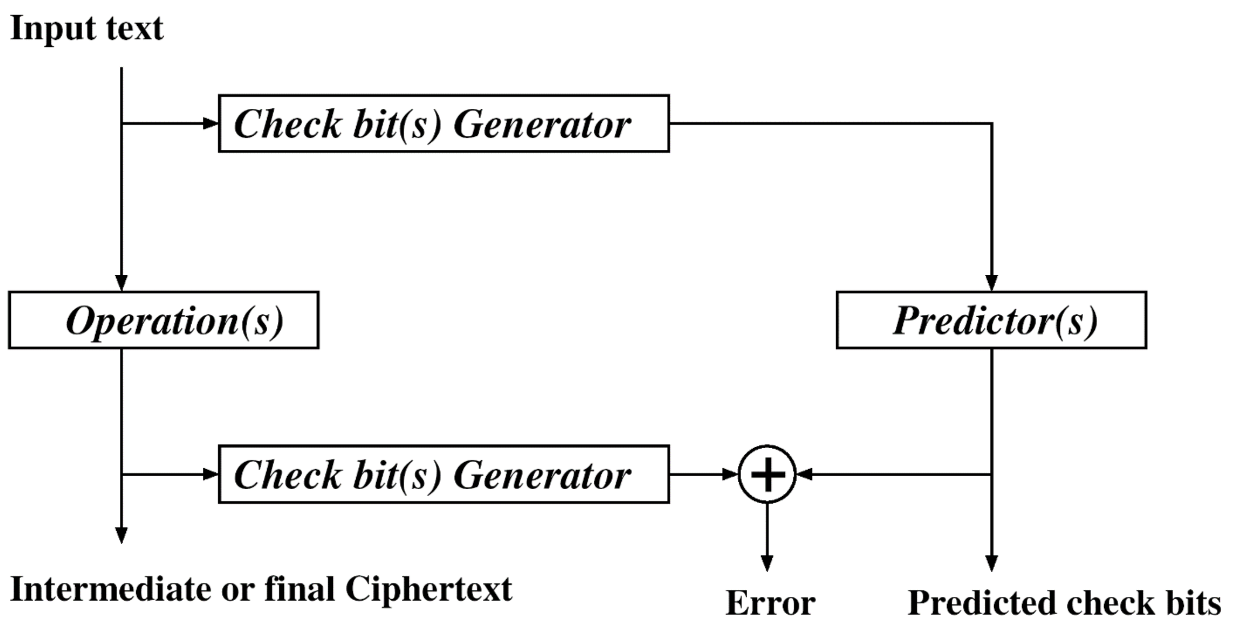
- **DDR**: Double data rate.— operate on both clock edge
- So you save clock cycles and use that for redundancy



Paulo Maistri and Regis Leveugle, IEEE TC 2008

Parity-based CED

- First generate check bits
- For each operation within encryption predict check bits
- Periodically compare predicted check bits to generated ones
- Predicting check bits for each operation - most complex step
  - Should be compared to duplication
- Can be applied at different levels –
  word, byte, nibble



Source : Koren and Krishna, Morgan-Kaufman 2007

Parity for AES

- Operations operate on bytes so **byte-level parity** is natural
- **ShiftRows:** Rotating the parity bits
- **AddRoundKey:** Add parity bits of state to those of key
- **SubBytes:** Expand Sbox to 256×9 – add output parity bit; to propagate incoming errors (rather than having to check) expand to 512×9 – put incorrect parity bit for inputs with incorrect parity
- **MixColumns:** The expressions are:

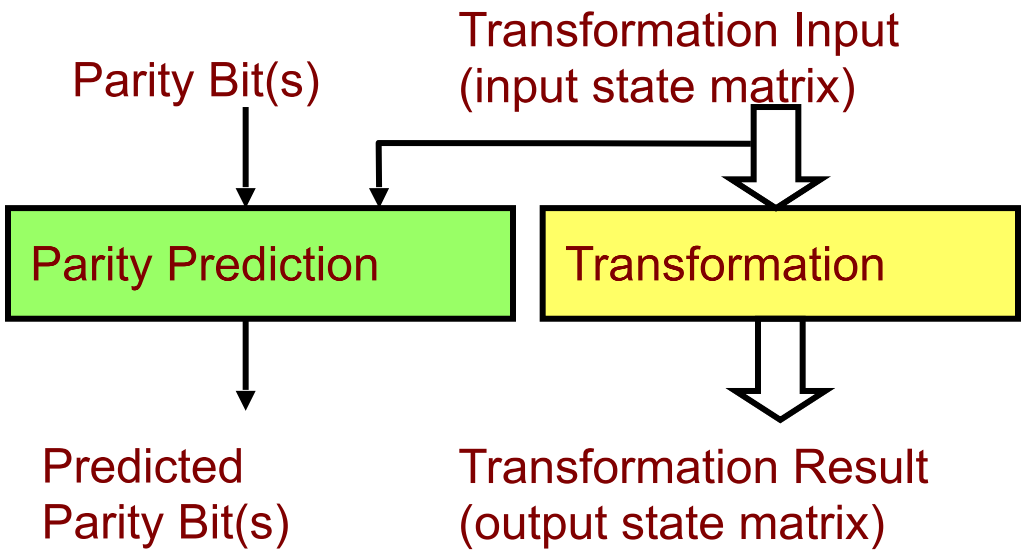
$$p_{0,j} = p_{0,j} \oplus p_{2,j} \oplus p_{3,j} \oplus S_{0,j}^{(7)} \oplus S_{1,j}^{(7)}$$

$$p_{1,j} = p_{0,j} \oplus p_{1,j} \oplus p_{3,j} \oplus S_{1,j}^{(7)} \oplus S_{2,j}^{(7)}$$

$$p_{2,j} = p_{0,j} \oplus p_{1,j} \oplus p_{2,j} \oplus S_{2,j}^{(7)} \oplus S_{3,j}^{(7)}$$

$$p_{3,j} = p_{1,j} \oplus p_{2,j} \oplus p_{3,j} \oplus S_{3,j}^{(7)} \oplus S_{0,j}^{(7)}$$

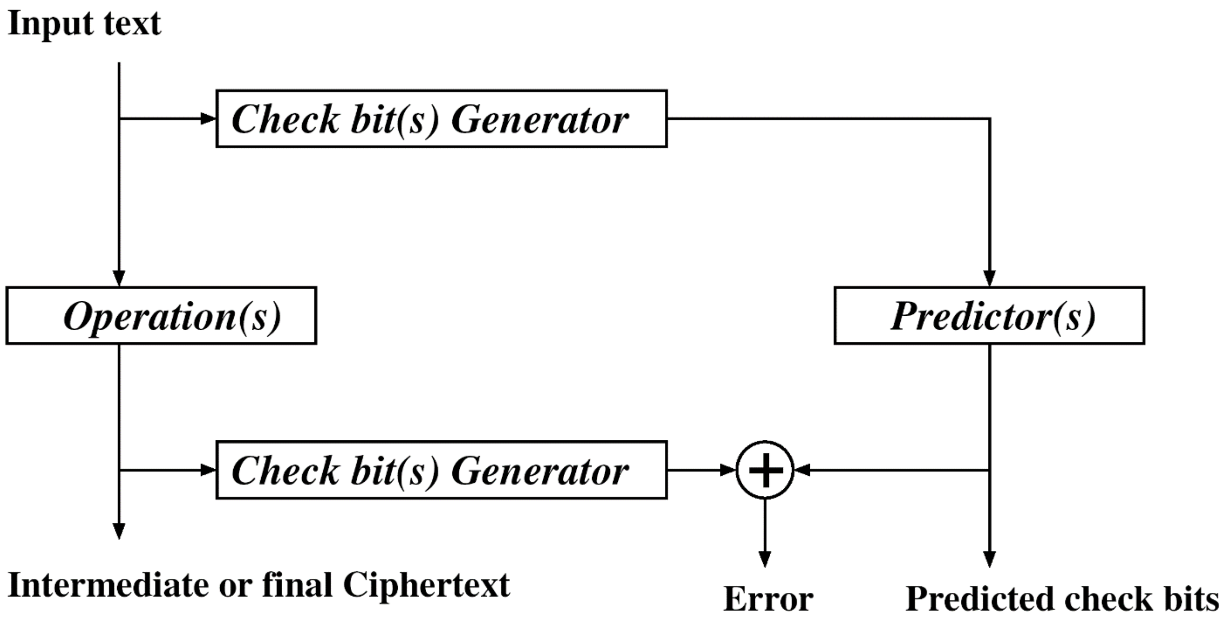
where $s_{i,j}^{(7)}$ is the msb of the state byte in position i,j



Source : Koren and Krishna, Morgan-Kaufman 2007

Parity-based CED

- Parity is just one example — can be other codes as well..
- Fault coverage is not great, all single-bit faults are detected
- There are some ways of using a **non-linear code** to improve fault coverage…
- Better linear codes can also be used with higher d.



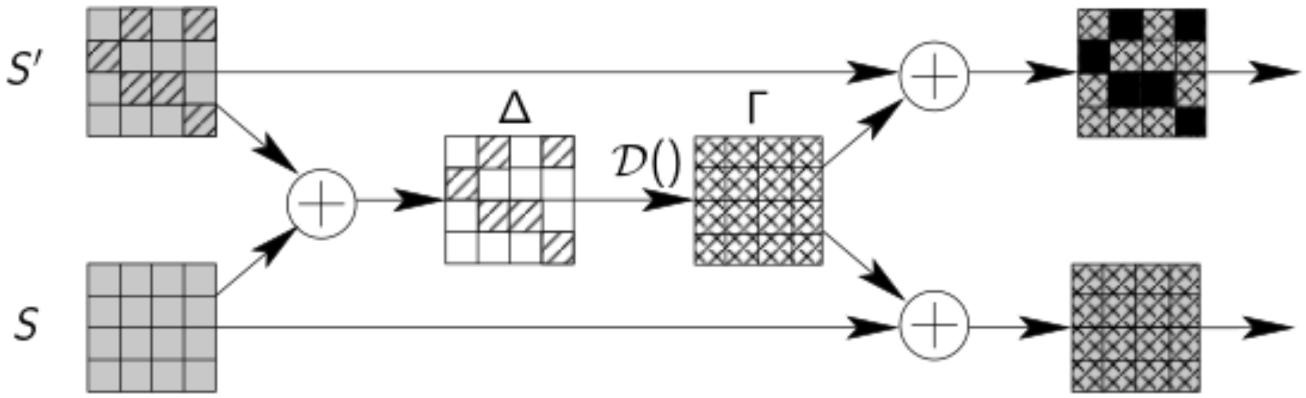
Source : Koren and Krishna, Morgan-Kaufman 2007

Infective Countermeasure

- I do not want to do the check…its just an if-else decision
- Someone can fault the check as well — so I **infect**…

Generic sketch exhibiting the Infection CM:

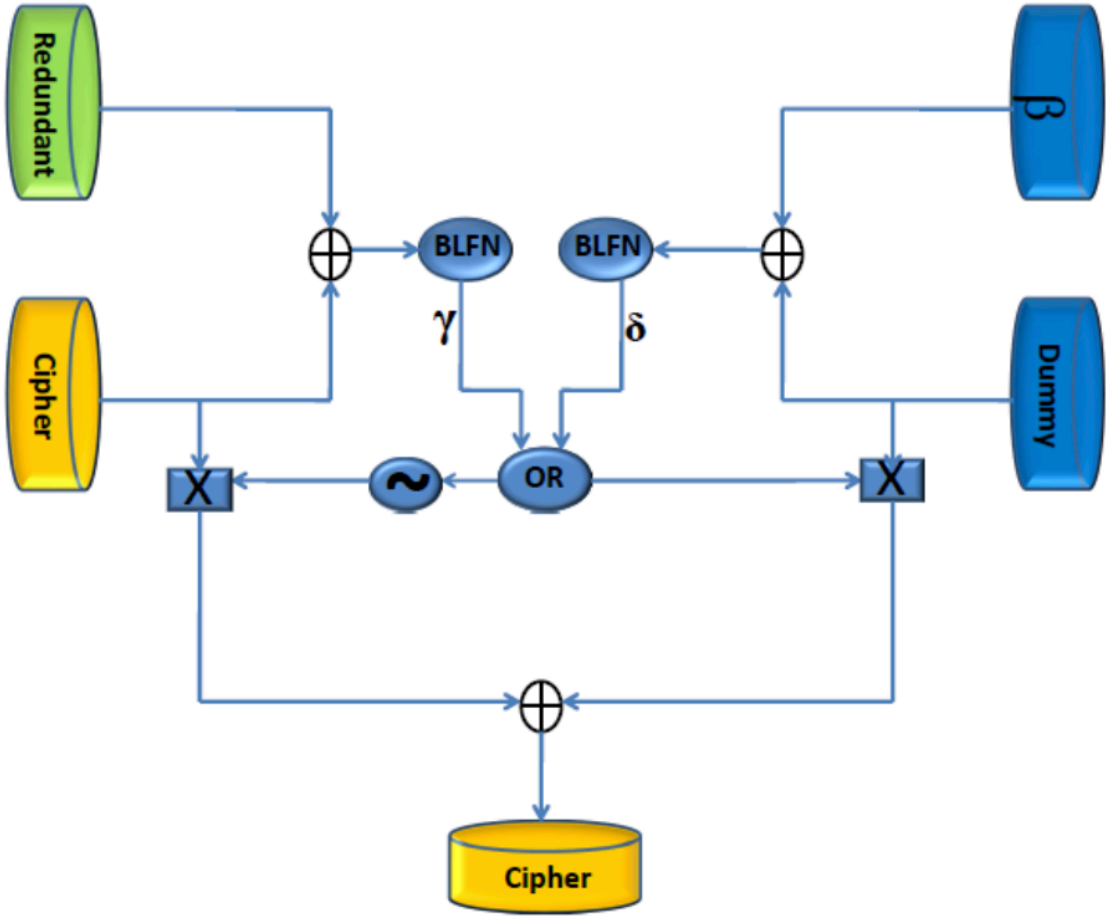
- S, S' the two States
- $\mathcal{D}$ the *diffusion function* (such as $\mathcal{D}(0) = 0$)



Source : Lomne et. al. , On the Need of Randomness in Fault attack Countermeasures – Application to AES, FDTC 2012

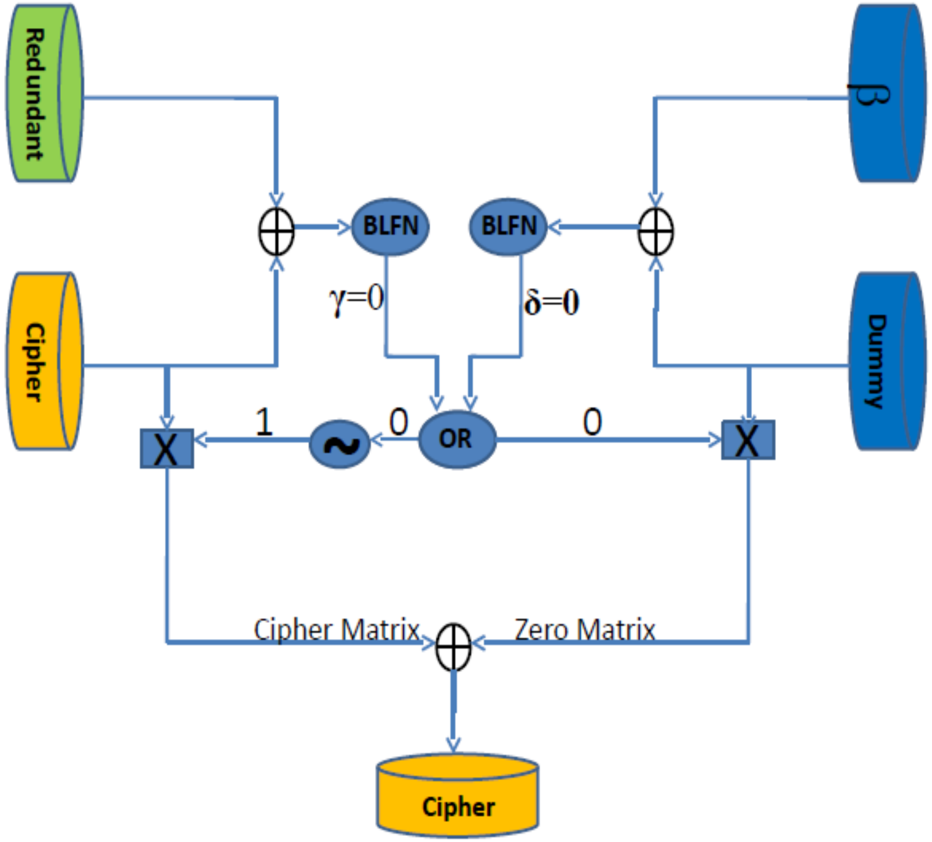
Infective Countermeasure

- I do not want to do the check…its just an if-else decision
- Someone can fault the check as well — so I **infect**…
- BLFN is a Boolean function with single bit output — has to somehow encoded to multi bits

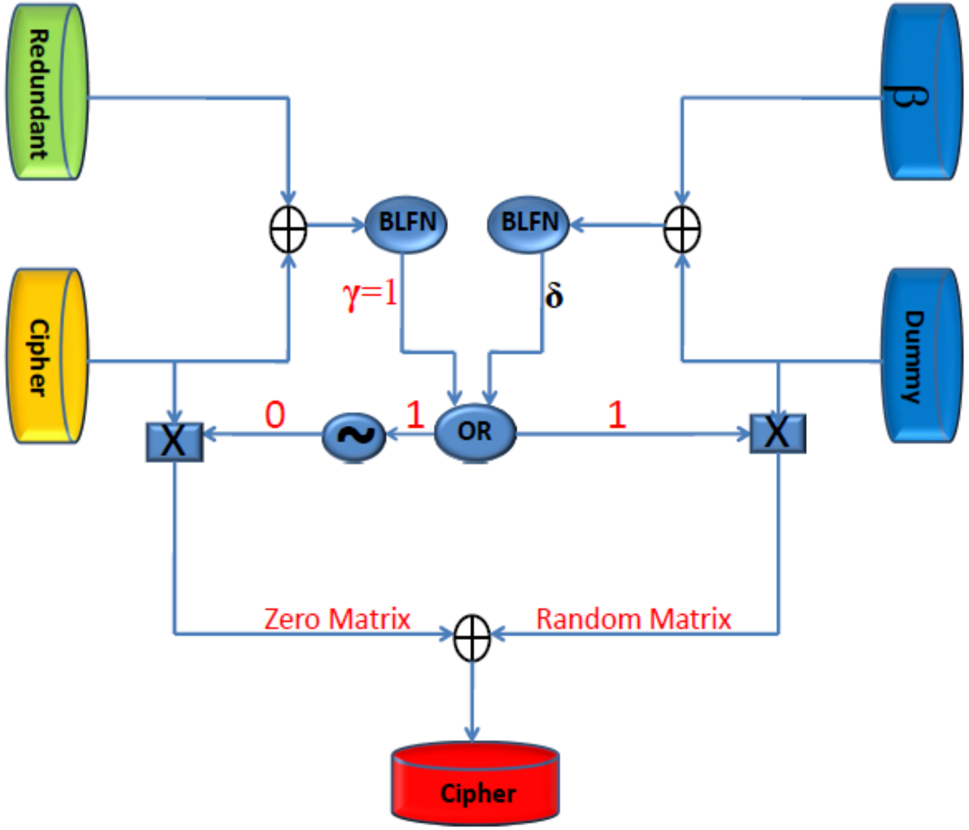


Infective Countermeasure

Correct Computation



Faulty Computation



Infective Countermeasure

Detection

- Detection countermeasures can be bypassed with biased faults.
  - **Patranabis et. al. at COSADE 2014.**
- The detection step leaks for random faults and combined side-channel.
  - **Saha et. al. in FDTC 2018.**

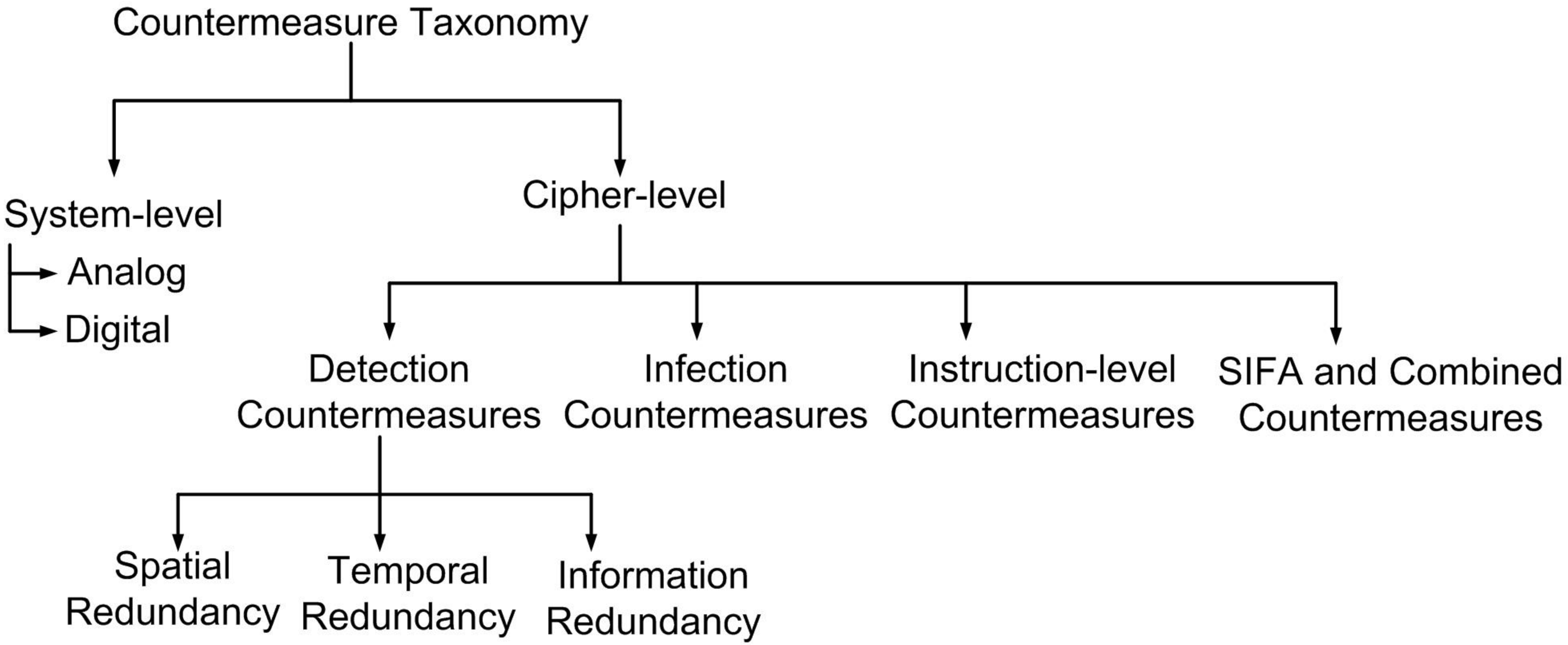
Infection

- Fournier et. al. and Joye et. al. : Deterministic infection function.
  - **Insecure [Lomne et. al. in FDTC 2012].**
- Gierlichs et. al. : Randomized infective countermeasure.
  - **Insecure [Battistello et. al. in FDTC 2013 ;Tupsamudre et. al. in CHES 2014].**
- Ghosh et. al. : Randomized infective countermeasure with non-linear mixing.
  - **Insecure [Banik et. al. INDOCRYPT 2015].**
- Fournier et. al. and Joye et. al. : Randomized infection function.
  - **Insecure against control faults [Patranabis et. al. in HASS 2017].**

Recent State

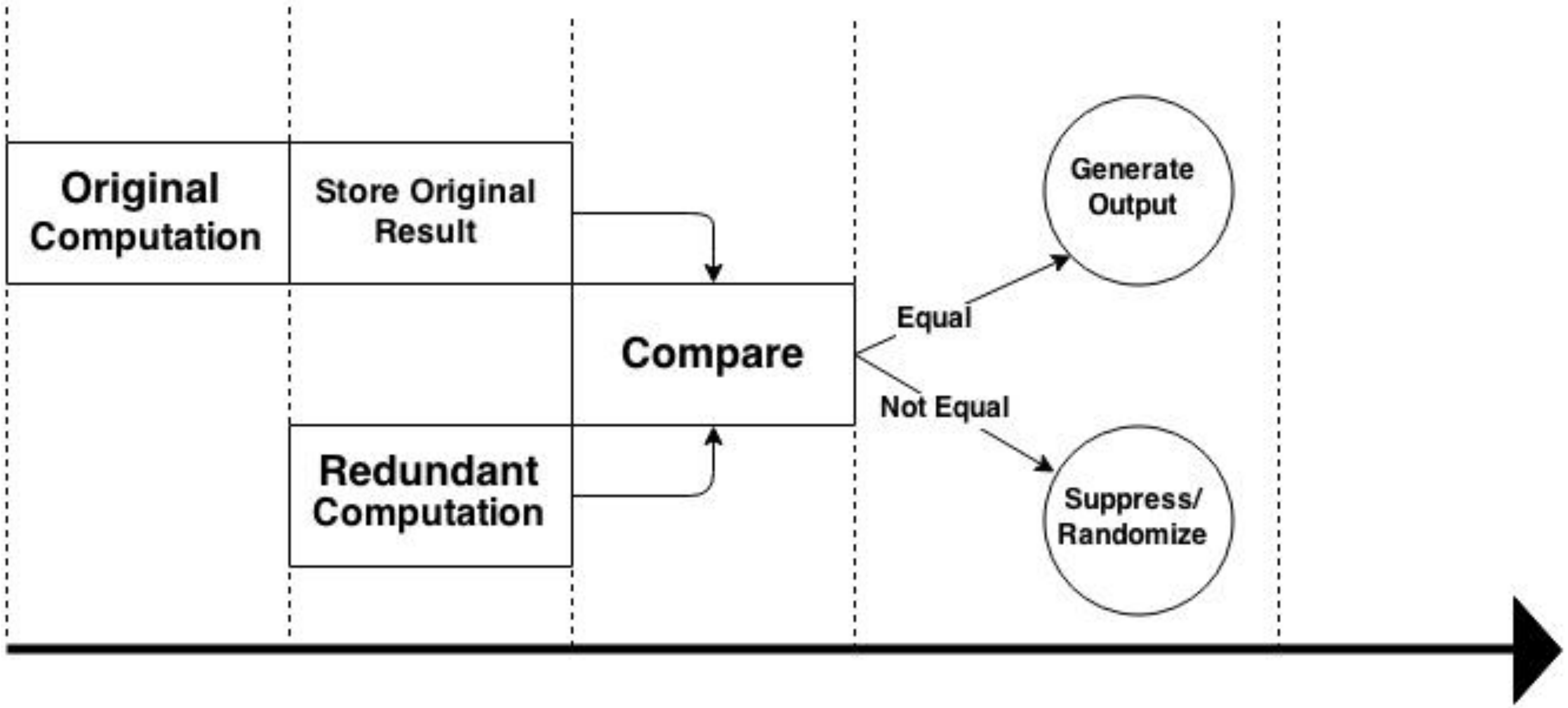
- **All the existing countermeasures are vulnerable against SIFA and FTA (invented by us).**

Countermeasures in Short…



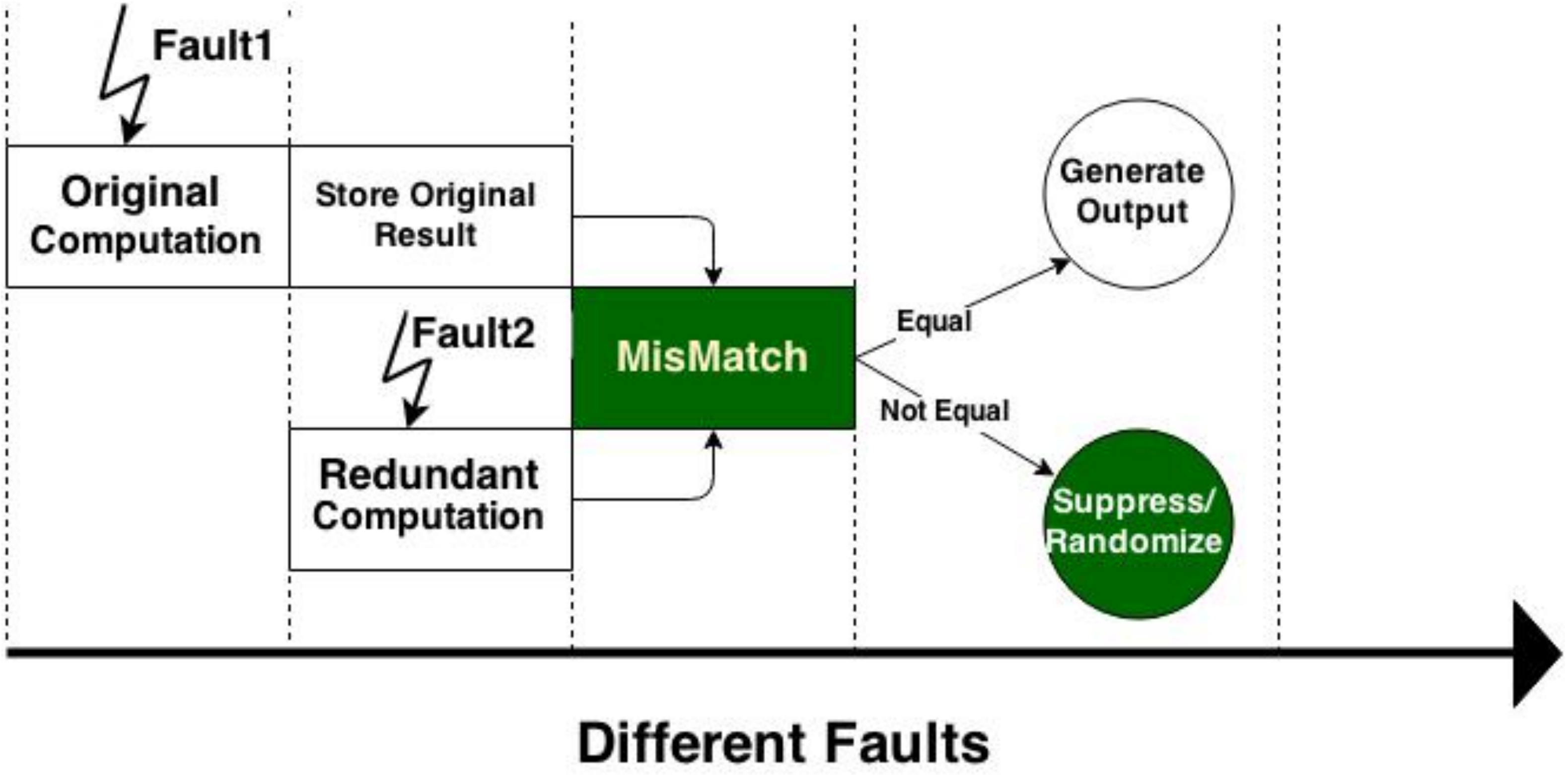
Let’s Break them All…

Attack 1: Time Redundancy Countermeasure

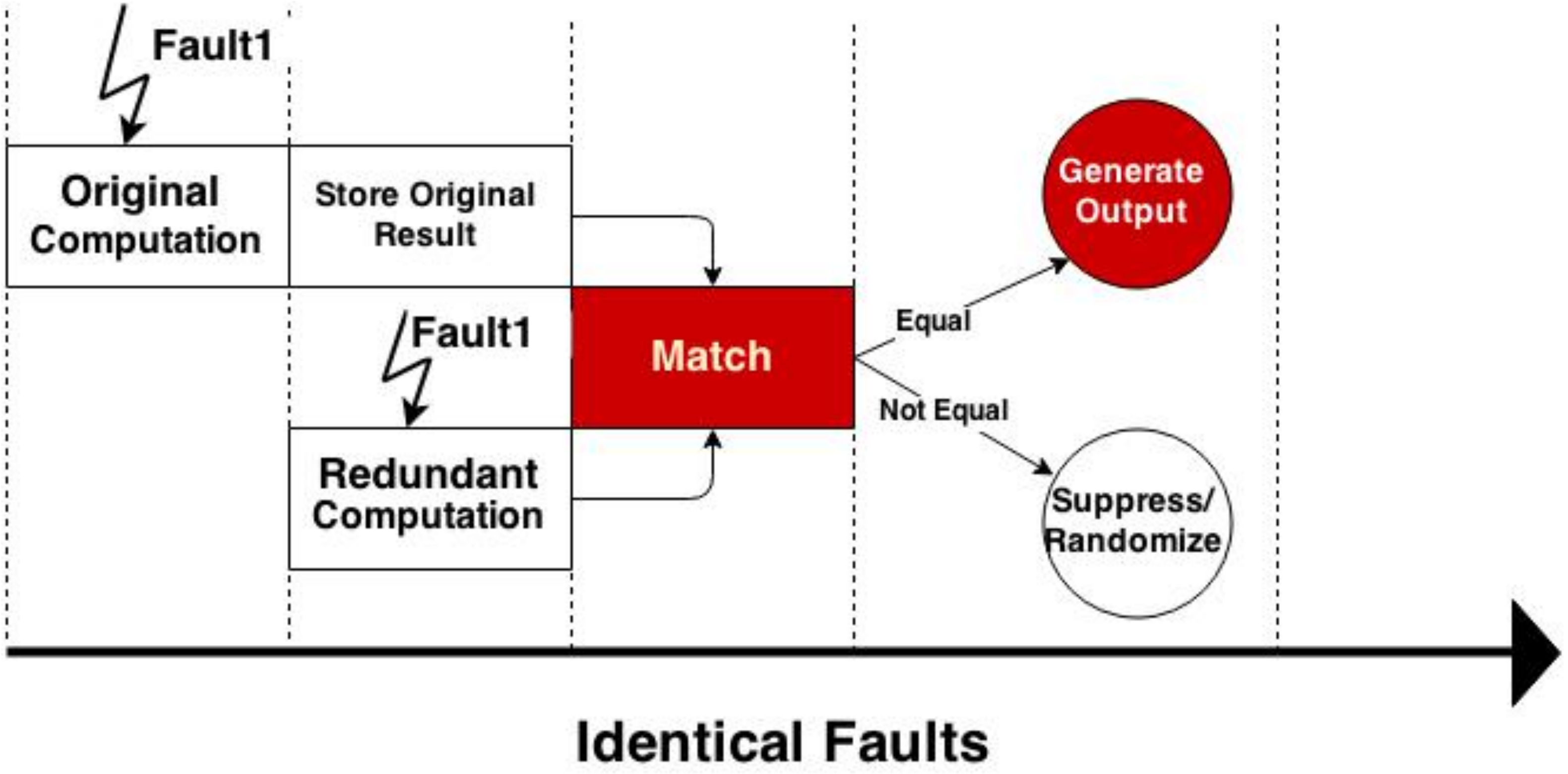


S.Patranabis, A.Chakraborty, P.H.Nguyen and D.Mukhopadhyay. A Biased Fault Attack on the Time Redundancy Countermeasure for AES. In *Proceedings of Constructive Side Channel Analysis and Secure Design 2015 (COSADE 2015*), Berlin, Germany, April 2015

Attack 1: Time Redundancy Countermeasure



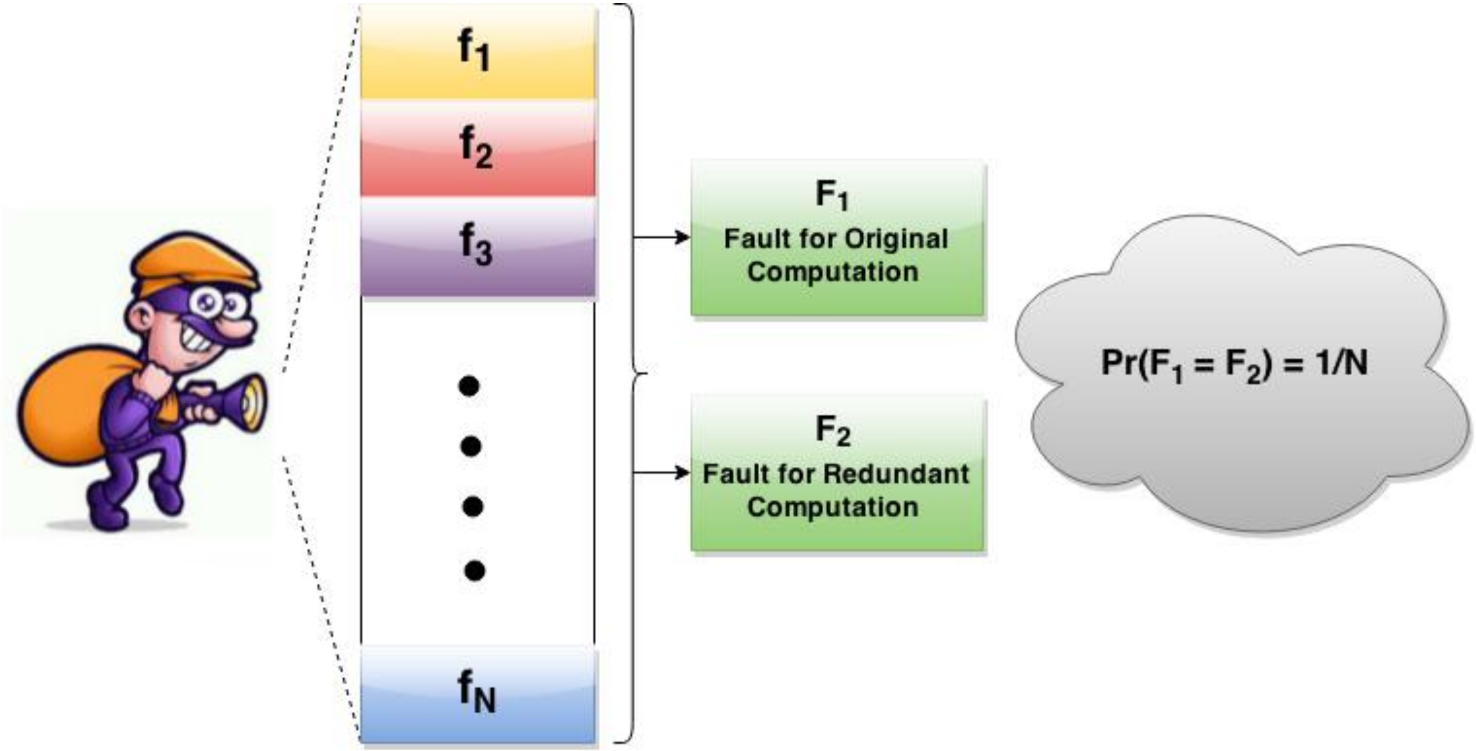
Attack 1: Time Redundancy Countermeasure



Attack 1: Time Redundancy Countermeasure

- Improving **fault collision probability**
  - Enhancing the probability of identical faults in original and redundant rounds
- Two major aspects
  - The size of the fault space
  - The probability distribution of faults in the fault space
- A smaller fault space enhances the fault collision probability
- A non-uniform probability distribution of faults in the fault space also enhances the fault collision probability

Attack 1: Uniform Random Faults



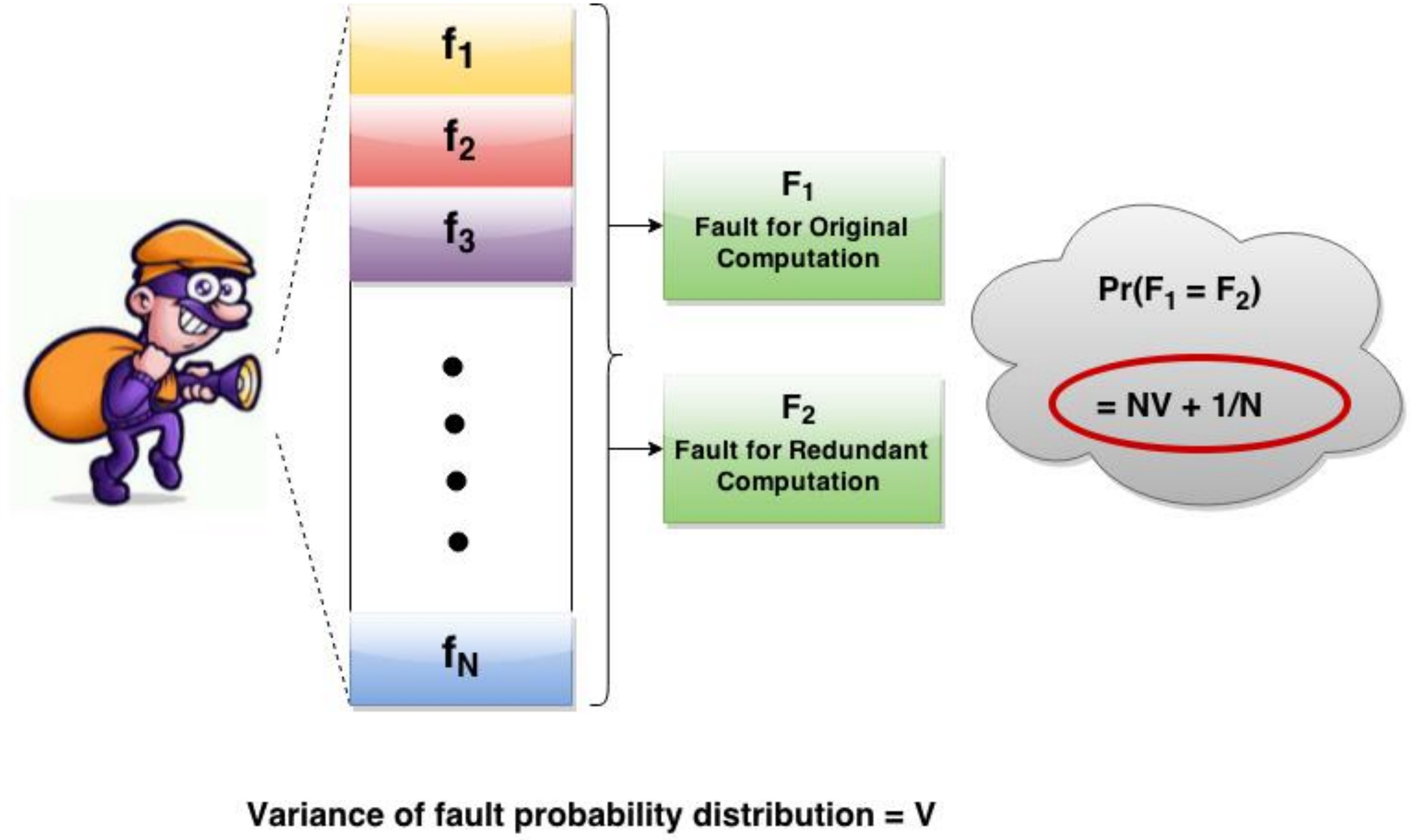
Attack 1: Biased Faults

- A total of n faults possible under a fault model F
- Each fault fi has a probability of occurrence Pr[fi]
- Let V be the variance of the fault probability distribution
- Degree of Bias of a fault model increases with increase in V

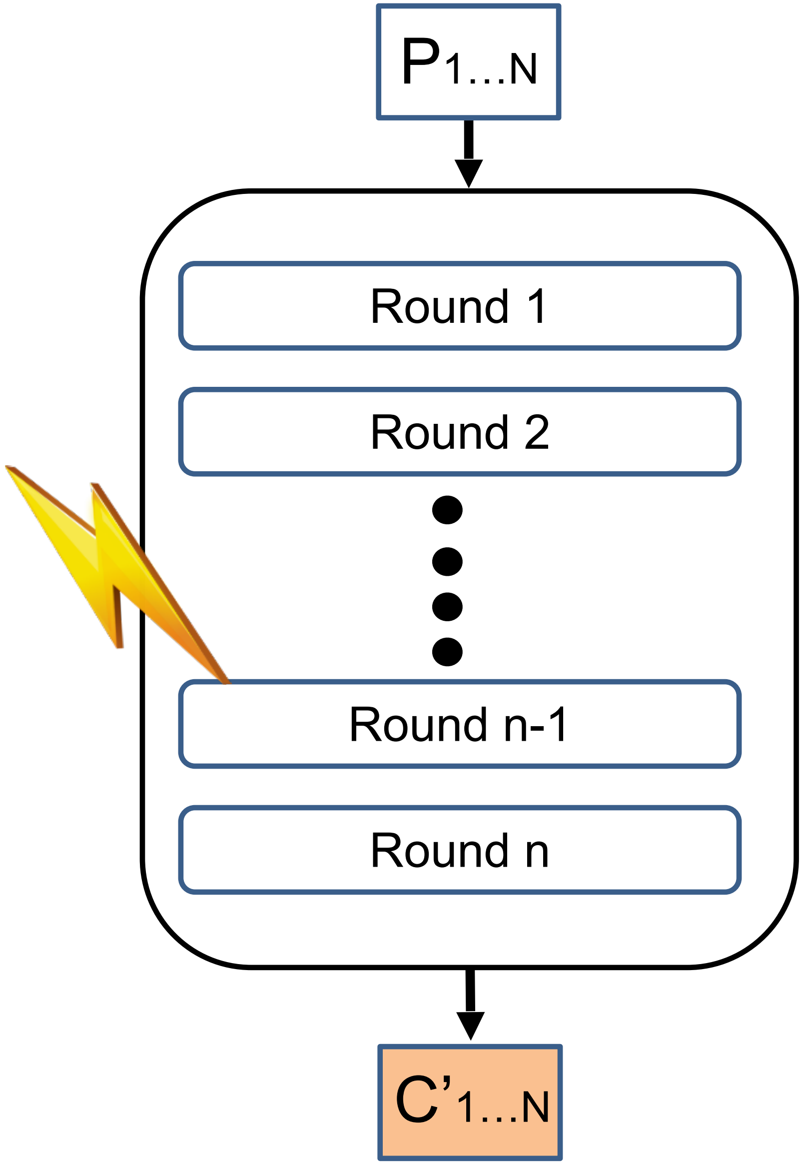
Fault Model	Pr[f1]	Pr[f2]	Pr[f3]	Pr[f4]	Pr[f5]	Pr[f6]	Pr[f7]	Pr[f8]	V
1	0.125	0.125	0.125	0.125	0.125	0.125	0.125	0.125	**0**
2	0.225	0.200	0.175	0.125	0.100	0.075	0.050	0.050	**0.004**
3	0.500	0.250	0.125	0.050	0.050	0.025	0	0	**0.026**

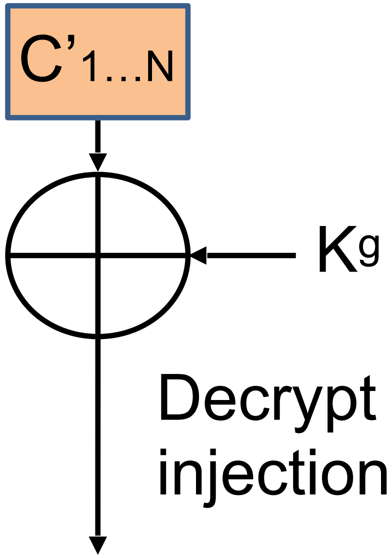
Attack 1: Biased Faults

- With increase in bias, collision probability increases
- Long-story short: you can get exploitable faulty ciphertexts with high probability, which you can use for attacks



Attack 1: With Statistical Fault Analysis (SFA)





Decrypt till the injection state

$$SEI(Kg) = \sum_{i=0}^{255}\left(\frac{\boxed{\#\{i \mid S^{-1}(C \oplus K_g) = i\}}}{n} - \frac{1}{256}\right)^2$$

Kg with highest SEI is the correct key

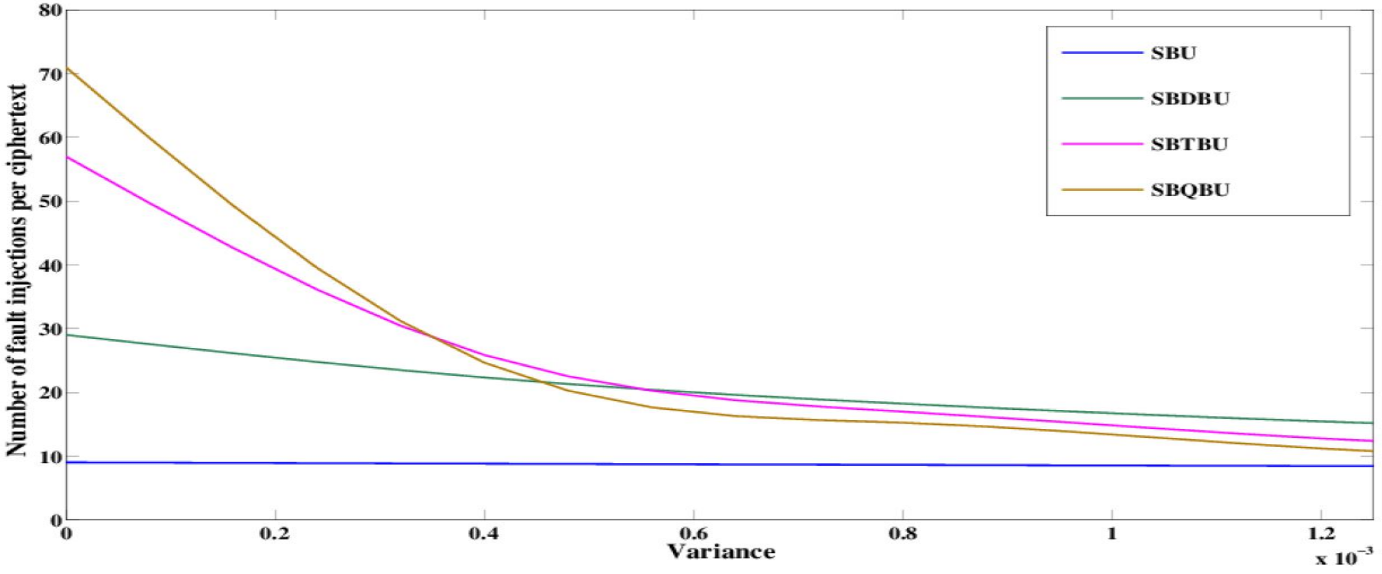
Attack 1: Biased Faults

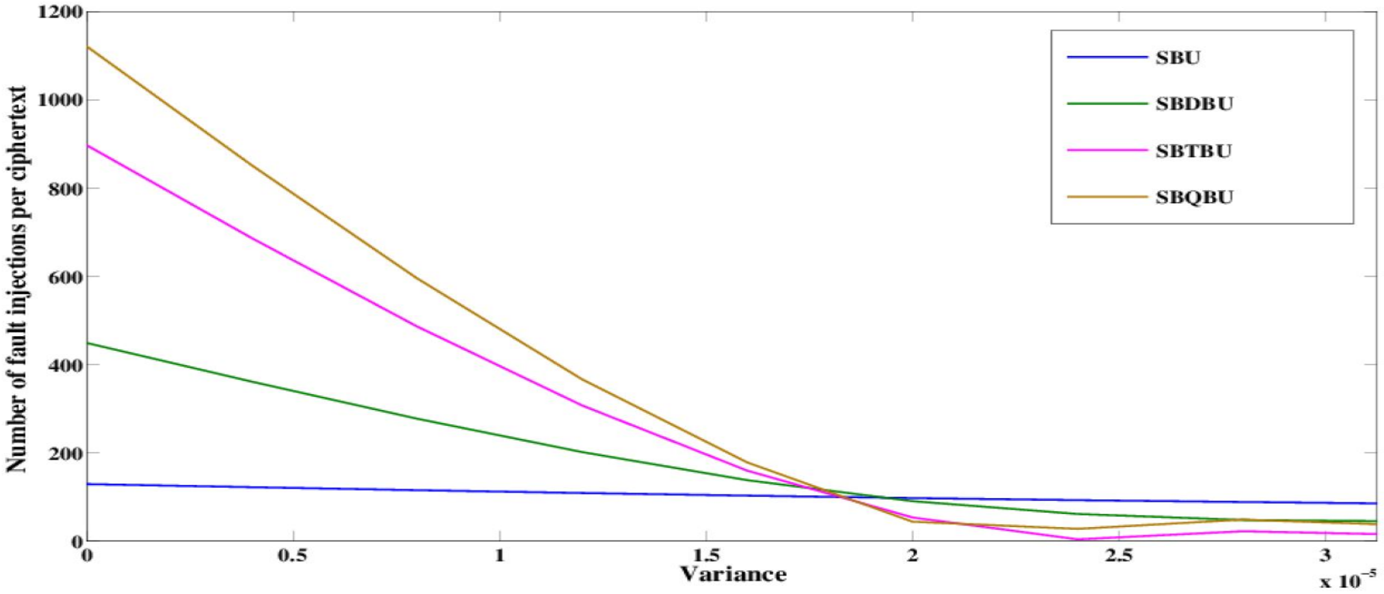
Useful ciphertexts

Total Fault Injections

Round	Fault Model	Fault Variance		N_C	N_F(simulation)		N_F(experimental)	
		Type-1	Type-2		Type-1	Type-2	Type-1	Type-2
8	SBU	9.5×10^{-2}	3.6×10^{-3}	304.75	340.48	647.52	387.67	687.91
	SBDBU	1.4×10^{-2}	9.2×10^{-4}	625.12	1456.25	1506.25	1448.45	1652.30
	SBTBU	9.7×10^{-3}	4.9×10^{-4}	1020.49	1815.60	2315.40	1974.86	2395.83
	SBQBU	3.2×10^{-3}	5.9×10^{-5}	1878.55	7868.82	28038.54	8003.14	30201.41
9	SBU	9.2×10^{-2}	3.5×10^{-3}	304.24	385.88	603.11	387.98	632.71
	SBDBU	8.8×10^{-2}	7.9×10^{-4}	624.65	641.18	1487.36	647.82	1556.69
	SBTBU	8.1×10^{-2}	6.7×10^{-4}	832.32	873.56	2054.00	878.23	2489.25
	SBQBU	7.5×10^{-2}	3.5×10^{-5}	1328.22	1788.84	17239.10	1809.25	20145.66

Attack 1: Biased Faults

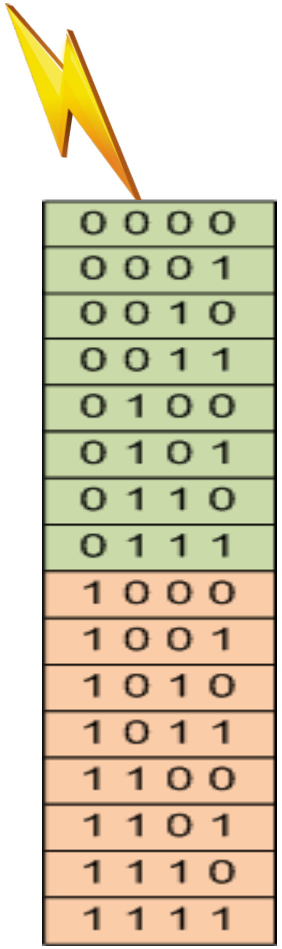


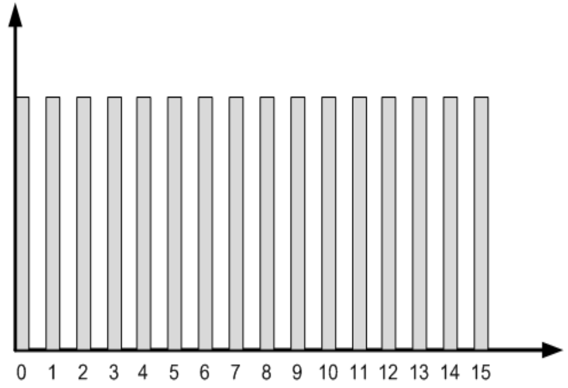


Let’s Break with a Single Fault…

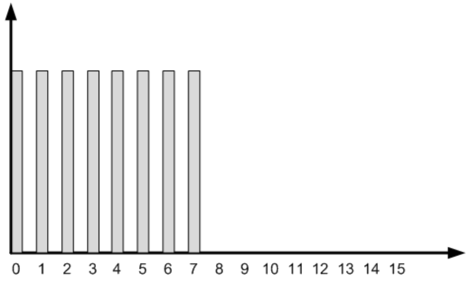
Statistical Ineffective Fault Analysis (SIFA)

- <u>The correct state space under the influence of faults is biased for biased faults.</u>
- <u>Example</u>:
  - Consider the stuck-at-0 fault at the MSB of a 4-bit state.
  - If value of the faulted state belongs to the first 8 table entries the ciphertext is correct.
  - The distribution of the state for correct values only assumes 8 possible values among 16.
  - Works for many other biased fault models.
- <u>SIFA utilizes correct ciphertexts for attack.</u>





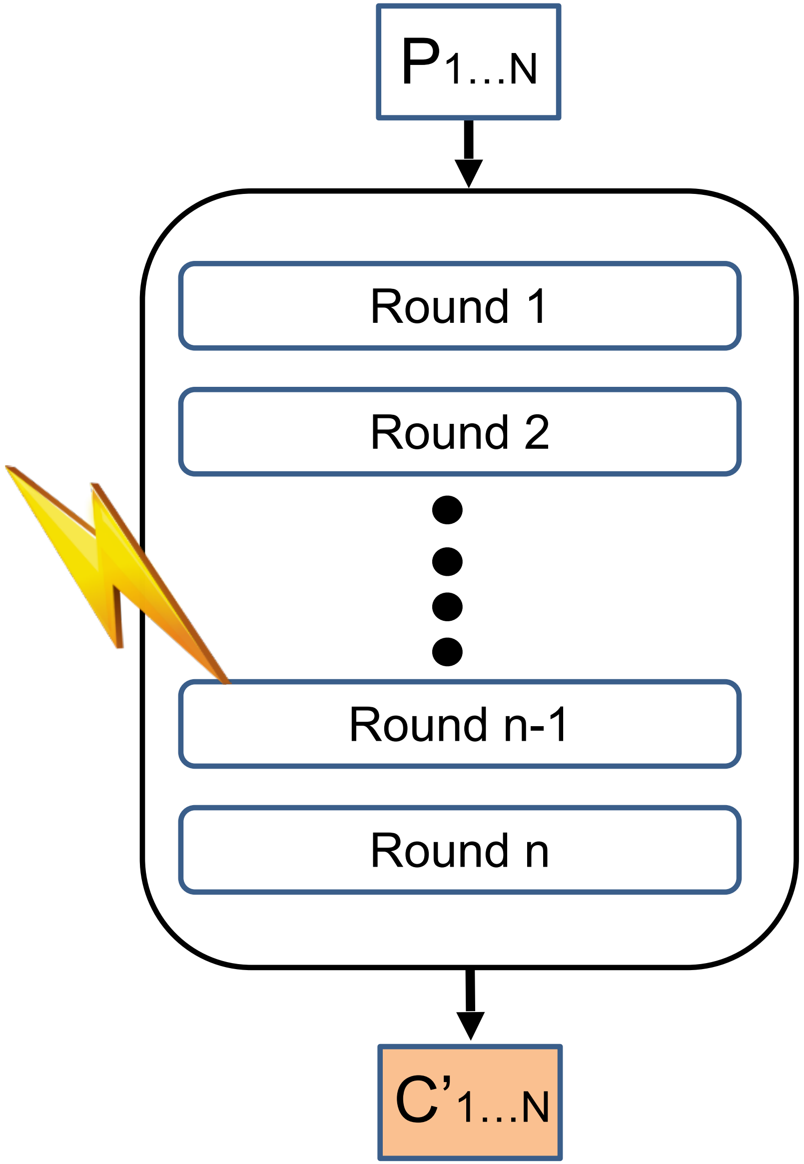
State distribution w/o faults

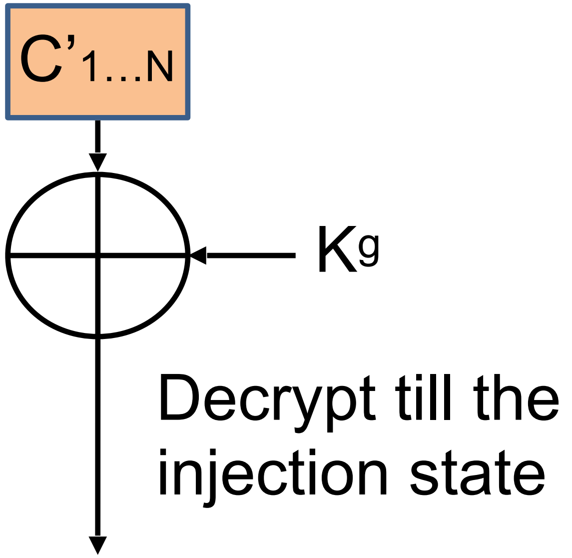


Correct State distribution with faults

C. Dobraunig, M. Eichlseder, T. Korak, S. Mangard, F. Mendel, R. Primas, "SIFA: Exploiting Ineffective Fault Inductions on Symmetric Cryptography". TCHES 2018

Attack 1: With Statistical Fault Analysis (SFA)



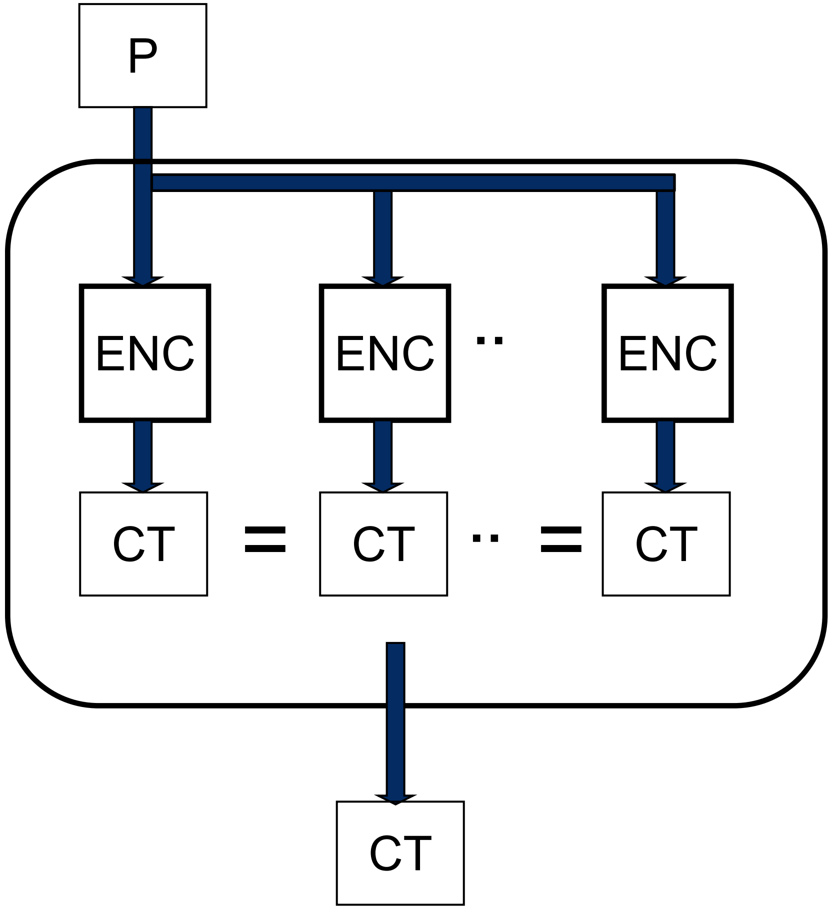


$$SEI(Kg) = \sum_{i=0}^{255} \left(\frac{\boxed{\#\{i \mid S^{-1}(C \oplus K_g) = i\}}}{n} - \frac{1}{256} \right)^2$$

Kg with highest SEI is the correct key

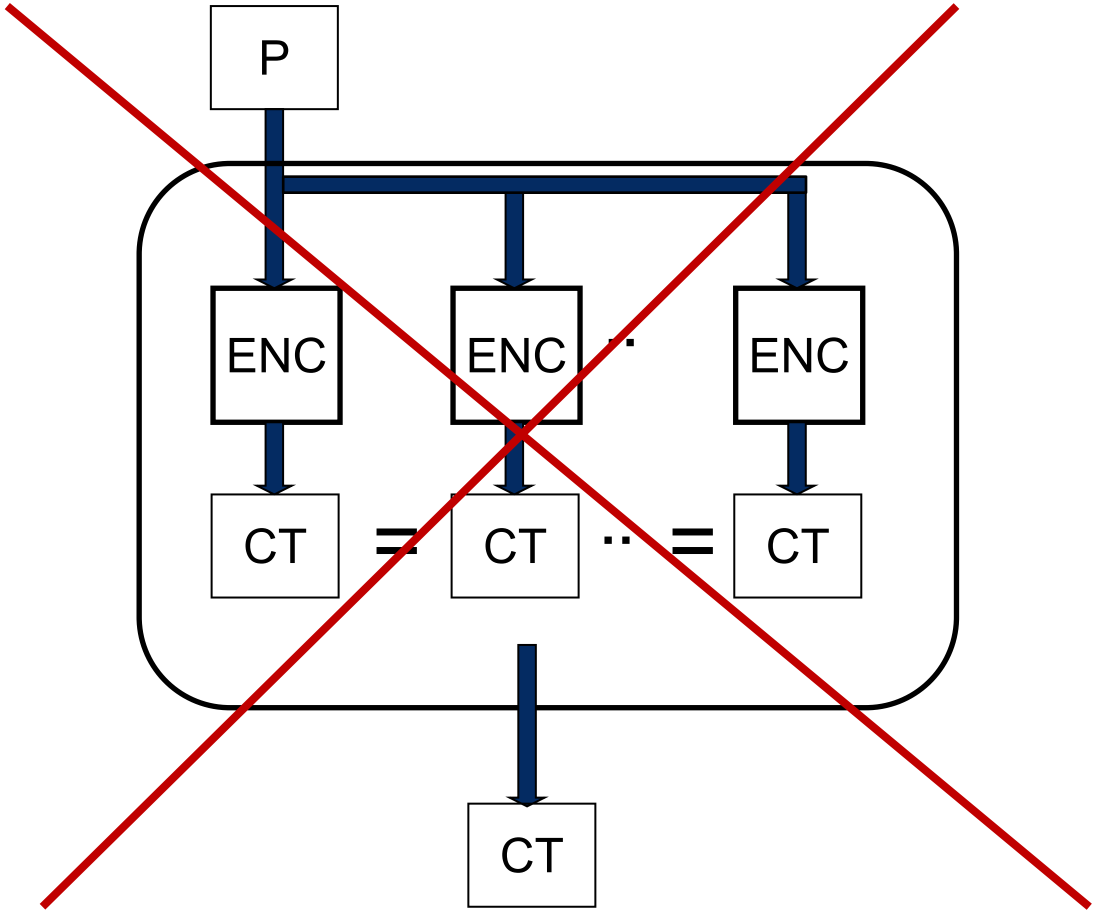
Why SIFA is Deadly?

- Most of the FA countermeasures mutes or randomizes the faulty ciphertexts. So, in many practical situations one can never get faulty ciphertexts with a single fault injection.
  - SIFA breaks this barrier.
  - Ineffective faults are feasible for both software and hardware implementations.
  - It is not typically limited to stuck-at faults.



Why SIFA is Deadly?

- Traditional detection/infection-based countermeasures are not useful.
- **Data-dependent Bias** in the faults is the main cause, which is the property of the fault. Maintaining an unbiased state in the presence of fault is tricky.



A Ray of Hope

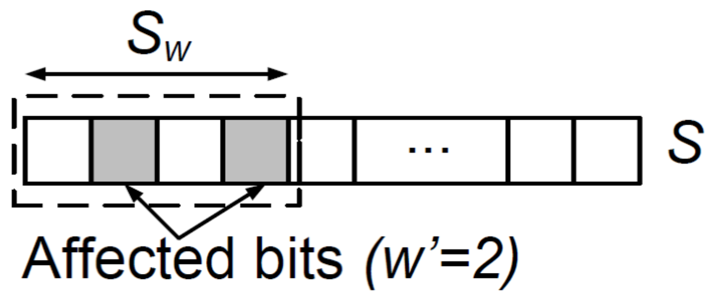
The Transform-and-Encode Framework

- A framework for realizing SIFA protection on block ciphers.
- Theoretically established security claims against SIFA.
- Uses state-of-the-art building blocks.
- Concrete realization on PRESENT called AntiSIFA.
- Practical security validation with laser fault injection.

The Transform-and-Encode Framework

Root cause of SIFA:

- Data-dependent bias due to fault injection



Ineffective Transition Probability

$$p^{i}_{x_i \to x_i}$$

Condition for SIFA

$$\exists\ x, y \in \mathcal{X}_{red} : p^{*}_{x}(x) \neq p^{*}_{y}(y)$$

Ineffective Transition Probability of a State

$$p^{*}_{x}(x) = \prod_{i=0}^{w'-1} p^{i_j}_{x_{i_j} \to x_{i_j}}$$

The Transform-and-Encode Framework

Condition for SIFA Prevention

$$\forall\ x, y \in \mathcal{X}_{red} : p_x^*(x) = p_y^*(y)$$

No SIFA

$x \backslash x'$	00	01	10	11
00	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$
01	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$
10	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$
11	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$

SIFA Happens

$x \backslash x'$	00	01	10	11
00	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$
01	$\frac{3}{8}$	$\frac{1}{8}$	$\frac{3}{8}$	$\frac{1}{8}$
10	$\frac{3}{8}$	$\frac{3}{8}$	$\frac{1}{8}$	$\frac{1}{8}$
11	$\frac{9}{16}$	$\frac{3}{16}$	$\frac{3}{16}$	$\frac{1}{16}$

The Transform-and-Encode Framework

Condition for SIFA Prevention

$$\forall\ x, y \in \mathcal{X}_{red} : p_x^*(x) = p_y^*(y)$$

No SIFA

$x \backslash x'$	00	01	10	11
00	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$
01	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$
10	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$
11	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$

SIFA Happens

$x \backslash x'$	00	01	10	11
00	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$
01	$\frac{3}{8}$	$\frac{1}{8}$	$\frac{3}{8}$	$\frac{1}{8}$
10	$\frac{3}{8}$	$\frac{3}{8}$	$\frac{1}{8}$	$\frac{1}{8}$
11	$\frac{9}{16}$	$\frac{3}{16}$	$\frac{3}{16}$	$\frac{1}{16}$

The Transform-and-Encode Framework

Randomized Domain Transform

- Parameterized security.
- SCA countermeasure masking is a *good* domain transform.
- Bypassing *n*-bit masking requires *n*-bit biased faults.

Algorithm Transform

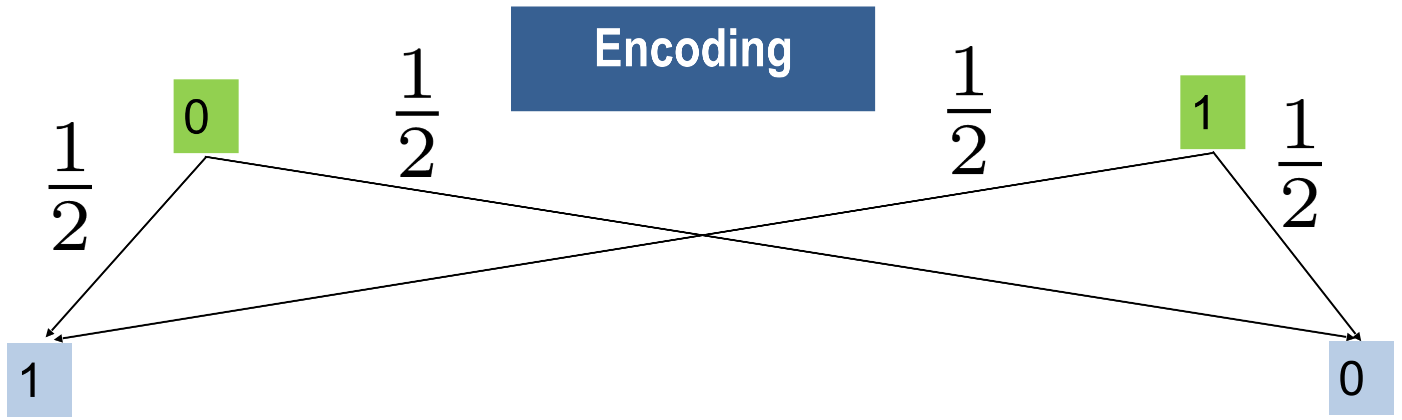
Input: $S \in \{0,1\}^n, d \in \mathbb{Z}$ where $d \leq n$
Output: $S^{tra} \in \{0,1\}^n, r \in \{0,1\}^d$
1: $[S_1 \ \cdots \ S_d] \xleftarrow{\lceil n/d \rceil} S$
2: $r \in_R \{0,1\}^d$
3: $S^{tra} \leftarrow \boxed{\text{DomTr}([S_1 \ \cdots \ S_d], r)}$
4: **Return** (S^{tra}, r)

Algorithm DomTr

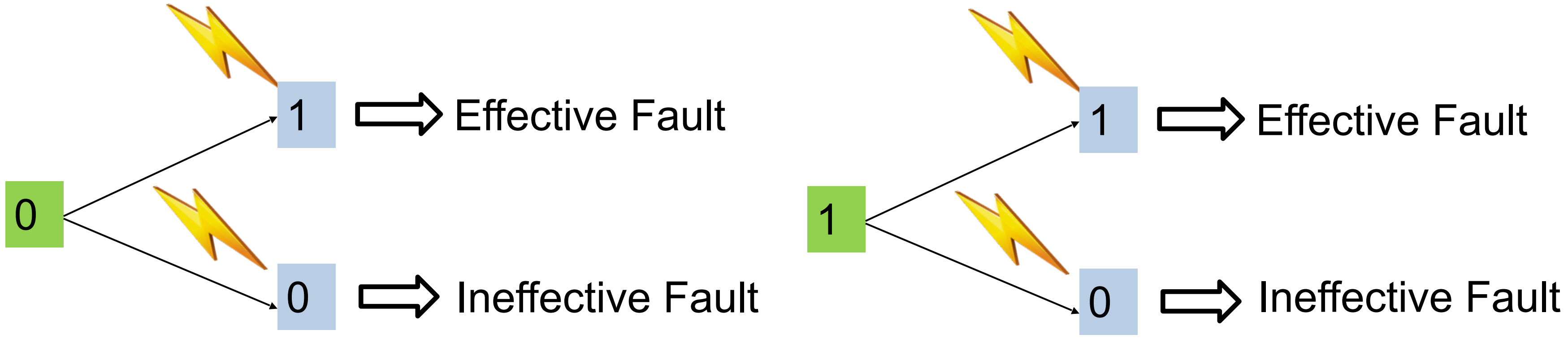
Input: $[S_1 \ \cdots \ S_d] \in \{0,1\}^n, r \in \{0,1\}^d$
Output: $S^{tra} \in \{0,1\}^n$
1: **for i = 1 to d**: $S_i^{tra} \leftarrow (r_i = 1) \ ? \ S_i \ : \ \overline{S_i}$
2: $S^{tra} \leftarrow [S_1^{tra} \ \cdots \ S_d^{tra}]$
3: **Return** S^{tra}

The Transform-and-Encode Framework

Transform: The Main Intuition



- Consider a stuck-at-0 fault
- **The fault in ineffective with probability 0.5 for both state value 0 and 1.**



The Transform-and-Encode Framework

The Transform Operation

d = 1

- **Generate a random bit *r*.**
- Complement the plaintext and entire encryption if $r = 1$.
- Protection against single-bit SIFA fault.

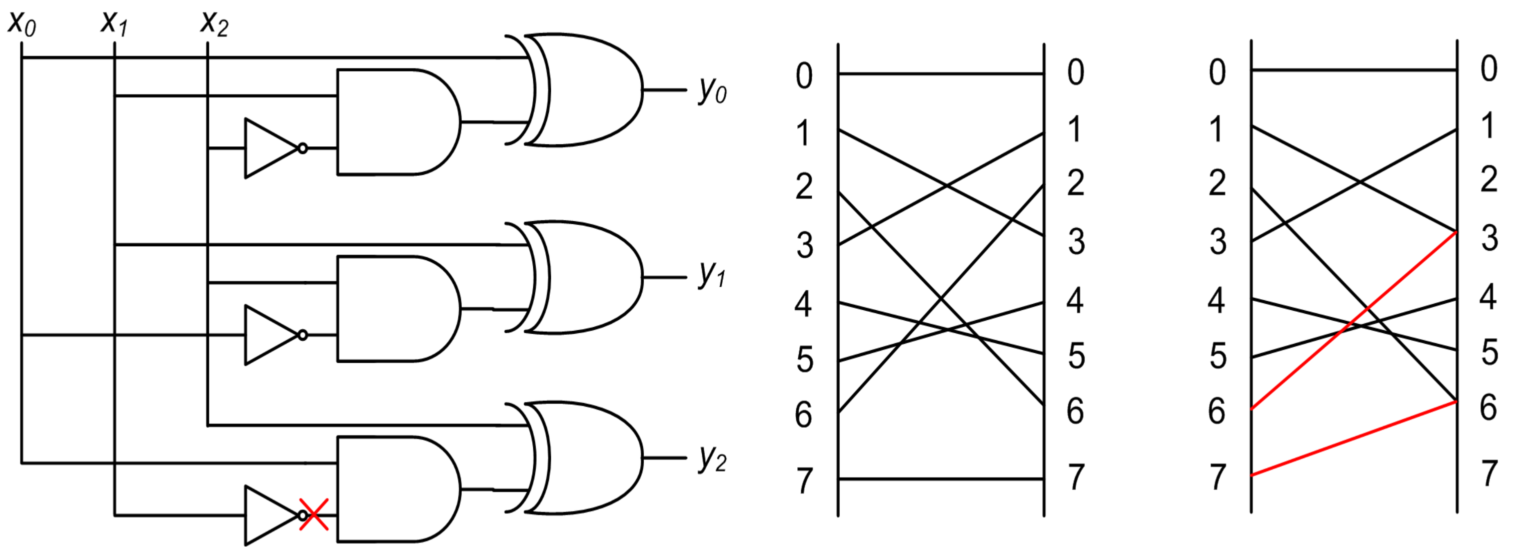
d = n

- **Generate one random bit *r* for each state bit.**
- Can be realized using **masking**.

The Transform-and-Encode Framework

SIFA on Masking:

- Statistical bias can be created even with unbiased bit flips.
- The fault has to be injected at the intermediate computation of an S-Box.
- Highly feasible for bit-sliced software implementations and masked hardware with intermediate registers.



C. Dobraunig, M. Eichlseder, H. Gross, S. Mangard, F. Mendel, R. Primas, “Statistical Ineffective Fault Attacks on Masked AES with Fault Countermeasures”. ASIACRYPT 2018

The Transform-and-Encode Framework

<u>Encode:</u>

- Error correction in a per-share level.
- Suggested after each non-linear layer, i.e. S-Box.
- Redundancy in correction operation.

Algorithm 4 Encode

Input: $S \in \{0,1\}^*, \gamma \in \mathbb{Z}$

Output: $S^{enc} \in \{0,1\}^{\gamma|S|}$

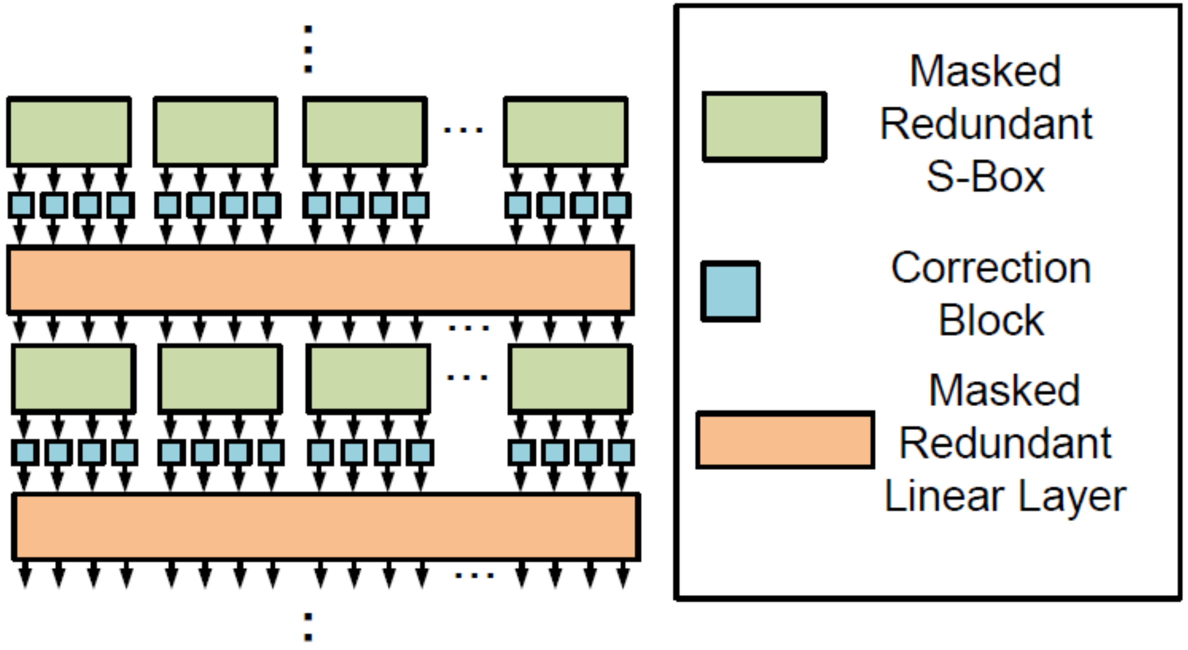
1: $S^{enc} \boxed{\leftarrow \text{ECC}(S, \gamma)}$

2: **Return** S^{enc}

AntiSIFA Countermeasure

<u>SIFA Countermeasure</u>

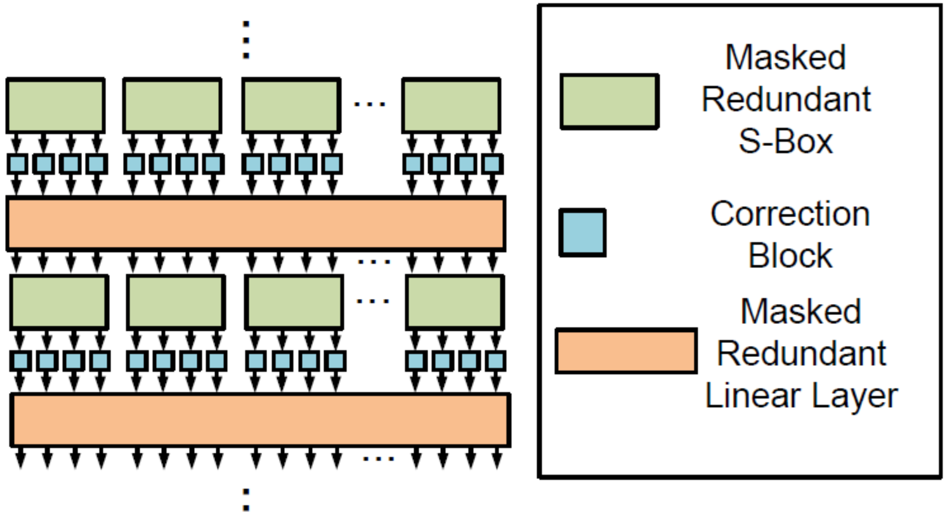
- Perform masking and then perform correction on each share
- This we can prove that this strategy prevents SIFA attacks
  - No ineffective faults
  - Masking alone prevents biased faults.
- This strategy is also effective against FTA.
- But it still does not end here…

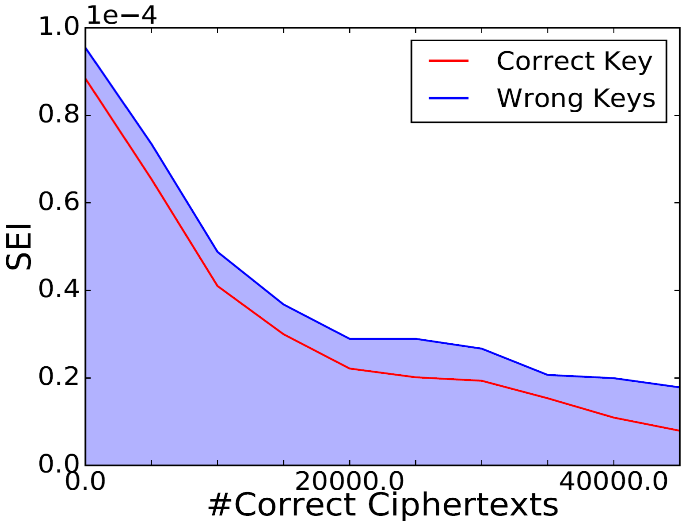


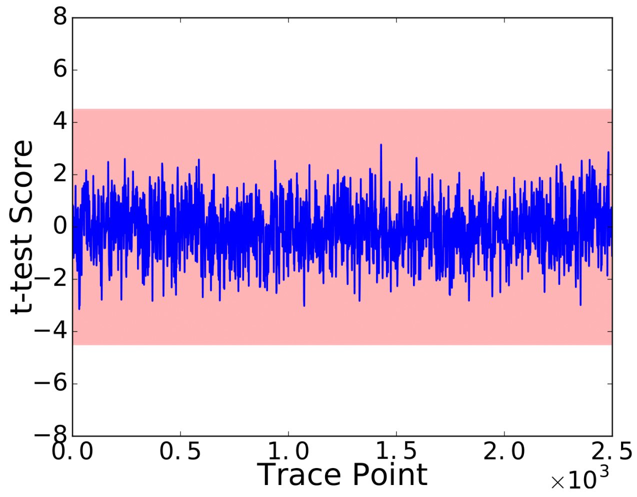
The Transform-and-Encode Framework

<u>AntiSIFA:</u>

- Transform-and-Encode for PRESENT
- First-order Threshold Implementation for masking
- Duplication code with $t = 1$ $(\gamma = 3)$ for error-correction.
- Ensures 3-bit security for fault injection outside S-Box (SIFA-1)
- Ensures 1-bit SIFA security for attack inside the S-Box (SIFA-2)
- Validated using laser faults.
- SCA security validated using TVLA.







To Conclude

- Fault attack countermeasure is still an partially unsolved problem
- TaE does not solve it entirely, but shows the first pathway
- Fault+Side channel combined attacks are possible — more deadly and breaks some variants of TaE.
- Fault Template Attack is another fundamental attack, which does not even need correct ciphertexts to attack. It can also attack middle rounds of any block cipher
- Recently, we have developed countermeasures against FTA and combined attacks— with cryptographic proofs
- But they are costly — we have still a long way to go…
- **Let the cat and mouse game continue…**